VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Maskování SQL Server a PostgreSQL databáze

SQL Server and PostgreSQL Database Masking

2018 David Prudič

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. David Prudič**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Maskování SQL Server a PostgreSQL databáze
SQL Server and PostgreSQL Database Masking

Jazyk vypracování: čeština

Zásady pro vypracování:

Pro maskování relační databáze existuje celá řada nástrojů, které umožňují maskovat cílovou databázi na základě definovaných kriterií. V rámci semestrálního projektu vznikl na Katedře informatiky prototyp nástroje, který podobné maskování umožňuje také.

Cílem této práce je rozšíření tohoto nástroje tak, aby splňoval následující požadavky:
1. Veškeré nastavení maskování hodnot a struktura databáze bude součástí projektu, který bude možné uložit. Aplikace bude řešit i možné nekonzistence mezi strukturou v projektu a v databázi.
2. Vznikne komponenta pro maskování hodnot, kterou bude možné použít samostatně (tzn. bez GUI aplikace pro definování maskování):
a) maskování databáze bude probíhat s pomocí hromadných operací,
b) nástroj bude podporovat SQL Server a PostgreSQL,
c) každý základní datový typ (řetězec, číslo, čas) bude možné maskovat s pomocí tří základních maskovacích metod.
Práce bude probíhat v následujících krocích:
1. Vytvoření unit testů pro stávající kód a provedení refaktoringu.
2. Rozšíření aplikace, aby splňovala výše uvedené požadavky.
3. Provedení funkčního a výkonnostního testování a dopracování aplikace s ohledem na výsledky testů.

Seznam doporučené odborné literatury:

[1] SQL Data Mask od RedGate, URL: https://www.red-gate.com/blog/audit-and-compliance/sql-data-mask
[2] Ravikumar, G. K., Manjunath, T. N., Ravindra, S. H., & Umesh, I. M. (2011). A survey on recent trends, process and development in data masking for testing. International Journal of Computer Science, 8(2).

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Radim Bača, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018 .....

Rád bych poděkoval panu Ing. Radimovi Bačovi, Ph.D. za vedení práce a vstřícnost při spolupráci. Děkuji také všem svým kolegům, kteří mi věnovali svůj čas, především panu Ing. Michalovi Paulechovi. A v neposlední řadě děkuji svým blízkým za podporu, bez které by tato práce nevznikla.

Abstrakt

Cílem diplomové práce je zlepšit již vytvořený nástroj pro maskování DuPE (Data ProtEctor). Zlepšení spočívá v implementaci podpory PostgreSQL databáze, imlpementaci API, které umožní použít aplikaci bez GUI, doplněním funkcionality, která bude řešit nekonzistence mezi uloženou strukturou databáze v projektu a samotnou databází a dále zvýšením výkonu maskování. Diplomová práce nejdřív popisuje problematiku maskovaní dat. Dále popisuje již vytvořený nástroj a změny, které bylo potřeba provést pro rozšíření aplikace. Poslední dvě části práce jsou věnovány implementaci nových funkcionalit a výkonnostnímu testování.

Klíčová slova: maskování databáze, bezečnost dat, PostgreSQL, SQL Server, c#, .NET

Abstract

The aim of the thesis is to improve already existing system DuPE (Data ProtEctor). The improvement lies in implementation of PostgreSQL support, implementation of an API, which will allow use application without GUI, adding functionality which will deal inconsistency between database structure saved in the project and the database itself and increase performance of data masking. The beginning of the thesis describes data masking problematic. The thesis further describes already existing tool and the changes needed to be done for extending the application. The last two parts of the thesis are dealing with the implementation of new functionalities and performance testing.

Key Words: data masking, data security, PostgreSQL, SQL Server, c#, .NET

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
CSV	–	Comma Separated Values
DAL	–	Data Access Layer
ER	–	Entity Relationship
GUI	–	Graphical User Interface
IDE	–	Integrated Development Environment
LA	–	Logical Access
ORM	–	Object Relational mapping
SQL	–	Structured Query Language
SSD	–	Solid State Drive
T-SQL	–	Transact-SQL
WPF	–	Windows Presentation Foundation
XML	–	Extensible Markup Language

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Data jsou v dnešní době pro většinu technologických společností alfou a omegou jejich podnikání. Je proto pochopitelné, že tyto společnosti vynakládají nemalé prostředky na jejich zabezpečení proti odcizení či ztrátě. I přes vysokou úroveň zabezpečení se stává, že k úniku dat dochází. Možností, jak k úniku dat může dojít, je mnoho, ale pro jednoduchost bychom je mohli rozdělit na dvě základní. Buď data byla ukradena, nebo k nim dostala přístup osoba, která je zneužila. Tou osobou může být klidně zaměstnanec, který vyvíjí rozšíření firemního systému. Nebo člověk z externí firmy, který pomáhá vylepšit fyzický návrh databáze. Takových osob může být nespočet. Dá se říci, že čím více osob má přístup k datům, která mohou být zneužita, tím je větší pravděpodobnost, že ke zneužití dojde.

Jedním ze způsobů, jak omezit počet osob s přístupem k citlivým datům, je poskytovat data pro testovací účely bez citlivých údajů. Ano, může být vznesena námitka, že testování na neostrých datech není dostačující. Ale tento problém řeší maskování dat, kterým se zabývá tato diplomová práce. Maskování dat totiž vytvoří na první pohled reálná data se stejným statistickým charakterem jako původní data a zároveň data učiní bezcenná pro zneužití.

Začátek práce popisuje možné způsoby maskování dat a komerční nástroje k tomu určené. V další části je popsáno řešení již vytvořeného nástroje pro maskování, na což navazuje popis doplnění řešení o jednotkové testy a průběh refactoringu. V posledních částech je popsána implementace rozšíření aplikace a výkonové testování.

2 Maskování dat

Maskování dat je jeden ze způsobů jak data ochránit před zneužitím. Ochrana spočívá v takzvané anonymizaci dat, při které se původní data nahrazují daty uměle vytvořenými. Uměle vytvořená data zachovávají integritu dat a mohou zachovávat i relace z původních dat. Díky těmto vlastnostem lze zamaskovaná data používat pro účely vývoje a testování aplikací. V případě, že je tato technika použita a data jsou zcizena, nedochází k žádné újmě, protože ukradená data jsou bezcenná.

Maskování lze provádět pomocí dvou základních způsobů. Buď deterministicky tzn., že pokud se budou maskovat jedny data vícekrát, jejich zamaskovaná podoba bude vždy stejná. Tato metoda může být v některých případech výhodná, například pokud jsou data použita pro testování, kde je výsledek operace porovnáván s očekávaným výsledkem. Další způsob vytváří data náhodně, takže pokud budou jedny data maskována vícekrát, jejich zamaskovaná podoba bude vždy jiná. Tento přístup lze označit za bezpečnější, protože zpětná rekonstrukce dat je téměř nemožná.

V obou výše zmíněných způsobech lze využít následující techniky maskování:

- Substitution - původní hodnota je nahrazena novou hodnotou, která je vytvořena nebo vybrána z kolekce dat.
- Shuffle – hodnoty záznamů se promíchají mezi jednotlivými záznamy.
- Insertion – do tabulky se vloží nové záznamy
- Join – hodnota záznamu je vytvořena spojením více hodnot z dat

2.1 Existující nástroje

Pro maskování dat již existuje celá řada komerčních nástrojů, pro názornost, co takové nástroje umožňují a jak se s nimi pracuje, je přidán krátký popis dvou vybraných.

2.1.1 Data Masker for SQL Server

Data Masker je software postavený na .NETu verze 4.0 a lze ho vyzkoušet v 30 denní trial verzi. Aplikace podporuje tyto verze SQL Serverů 2000, 2005, 2008, 2012 a 2014.[1] Verze pro Oracle (Data Masker for Oracle) podporuje Oracle databáze těchto verzí 9i, 10g, 11g a 12c.[2]

Po spuštění programu je třeba vytvořit nový „Masking set“ nebo otevřít již vytvořený, „Masking set“ lze přirovnat například k projektu v IDE. „Masking set“ v sobě musí obsahovat alespoň jeden „Controller“ (což je připojení k databázi) a ten již obsahuje pravidla pro maskování („Rule“). Každé pravidlo musí mít jednoho rodiče a to právě již zmíněný „Controller“. Pravidla maskování se přidávají přes stromové zobrazení databáze pomocí kontextového menu.

Data Masker má několik druhů pravidel:

Maskovací pravidla Maskovací pravidla obsahují základní funkce pro maskování:

- Substitution: nahradí hodnoty jinými z daného zdroje
- Shuffle: záznamům ponechá primární klíč a promíchá dané hodnoty
- Insertion: vloží nové záznamy do tabulky
- Search-Repalce: umožnuje vyhledat řetězce a ty zaměnit za jiné
- XML Masker: umožnuje pracovat s xml uloženém v textu, např. vkládání hodnot do daných elementů z různých zdrojů

Synchronizační pravidla Se využívají, když více atributů sdílí stejná data např. jméno, příjmení a celé jméno. Tyto pravidla umožňují, skládat data z již vyplněných dat dané tabulky.

Speciální funkce Ve speciálních funkcích můžeme nadefinovat a spouštět T-SQL funkce.

Užitečné pravidla Pomocí užitečných pravidel můžeme vypínat a zapínat cizí klíče a trigry.

DataSety Data Masker obsahuje velké množství data setů od náhodných čísel přes fiktivní čísla bankovních účtů až po různé kombinace a v případě, že se nenajde vyhovující, tak si lze vyhovující vytvořit v textovém souboru a ten poté použít.

Dále umožnuje více vláknový běh a lze řídit i pořadí spouštění jednotlivých akcí (pravidel).

Data Masker je nástroj s nepřeberným množstvím možností a je určen spíše pro zkušenější uživatele a rozsáhlejší projekty.

Plusy:

- Velké množství možností nastavení maskování.
- Možnost zvolení pořadí jak se mají jednotlivé maskovací procesy provést.
- Velice dobrá dokumentace, tutoriály a podpora.

Mínusy:

- Pro jednoduché projekty zbytečně složité.
- 30 denní trial verze.
- Zvlášť verze pro MS Sql Server a Oracle db.

2.1.2 DataVeil

Je software postavený na Javě verze 7, který má základní bezplatnou verzi. Program podporuje tyto verze MS Sql Serverů 2008, 2012, 2014, 2016 a Oracle databáze těchto verzí 10g, 11g a 12c.[3] Stejně jako v předchozím softwaru je nejdříve třeba nastavit připojení k databázi, těchto připojení může být více stejně jako v předešlém případě. Po tomto kroku je třeba vybrat jednotlivé atributy, které chceme maskovat, což lze provést několika způsoby: výběrem atributů v diagramu, výběrem atributů ve „stromě“ a nejzajímavějším způsobem automatickým vyhledáním atributů vhodných pro maskování. Do automatického vyhledání si může uživatel nastavit i vlastní podmínky, čímž se celý proces může zautomatizovat.

Maskovací funkce DataVeil obsahuje větší množství maskovacích funkcí, které se většinou dělí podle typu atributu např. adresa nebo jméno, u takovýchto typů je potom odpovídající nastavení. Dále je zde maskovací funkce pro vložení hodnoty pomocí SQL např. pro vložení celého jména složeného ze jména a příjmení. Také jsou zde funkce Shuffle pro promíchání záznamů. U každé maskovací funkce lze nastavit skupinu záznamů, pro které se má použít stejná hodnota, dále lze nastavit automatickou synchronizaci s dalšími atributy v tabulce nebo i tabulkách.

DataSety DataVeil podporuje i správu vlastních zdrojů dat vedených v csv souborech.

Mezi další užiteční funkce patří:

- Automatické seřazení maskovacích funkcí k provedení tak, aby nedošlo k chybě v důsledku závislostí.
- Automatické vypínání a zapínání trigrů a cizích klíčů.
- Možnost prohlédnout si data ve stavu před maskováním a po v jednom okně.

DataVeil se snaží být intuitivní a jednoduchý nástroj, který obstará co nejvíce práce za uživatele. Je určen spíše pro jednoduché projekty (s ohledem na dobu potřebnou pro ovládnutí programu).

Plusy:

- Zobrazení dat před a po maskování, zobrazení diagramu databáze.
- Automatické vyhledání citlivých dat vhodných pro maskování.
- Jedna verze pro MS Sql Server a Oracle db.
- Možnost spouštění přes cmd

Mínusy:

- Některé zautomatizované funkce nelze ovlivnit.
- Free verze má omezený počet použití určitých maskovacích funkcí.

3 Popis stávajícího řešení

Stávající řešení je postaveno na platformě .NET 4.0 v programovacím jazyce C# s využitím grafické knihovny WPF, tato volba umožňuje aplikaci spustit na operačním systému Windows verze 7 až 10. Další použité knihovny jsou log4net na logování chování aplikace a nHibernate na ORM. Jako vývojové prostředí bylo použito Microsoft Visual Studio Enterprise 2017.

3.1 Popis použitých technologií:

.NET

.NET Framework je vývojová platforma pro vytváření aplikací pro web, Windows, Windows Phone, Windows Server a Microsoft Azure, která poskytuje mnoho služeb, včetně správy paměti, zabezpečení, práce se sítí a zavádění aplikací. Poskytuje snadno použitelné datové struktury a rozhraní API, které abstraktně překračují operační systém Windows. Pro .NET Framework lze použít různé programovací jazyky včetně C#, F# a Visual Basic. Základní komponentou je Microsoft .NET Framework, prostředí potřebné pro běh aplikací a nabízející jak spouštěcí rozhraní, tak potřebné knihovny. Pro vývoj .NET aplikací vydal Microsoft Visual Studio .NET.[4]

WPF

Windows Presentation Foundation (WPF, dříve Avalon) je v knihovna tříd pro tvorbu grafického rozhraní, která je součástí .NET frameworku od verze 3.0 firmy Microsoft (je nástupcem Windows Forms). WPF je součástí Windows Vista, Windows 7 a Windows Server 2008 a je možné ji doinstalovat do Windows XP SP2/SP3 a Windows Server 2003. Pro vytvoření „uživatelsky bohatého rozhraní“ (RUI) využívá značkovací jazyk XAML, který umožňuje oddělit funkčnost a vzhled aplikace. Cílem WPF je sjednotit uživatelské rozhraní, 2D a 3D grafiku, vektorovou a rastrovou grafiku, animace a provázat s daty audia a videa.[5]

NHibernate

NHibernate je nástroj pro objektově relační mapování (ORM) pro platformu .NET. Smyslem NHibernate je ulehčit vývojářům práci při řešení úloh týkajících se perzistence dat.[6]

log4net

Knihovna log4net je nástroj, který slouží k zpracování logovacích informací do více možných formátů výstupu.[7]

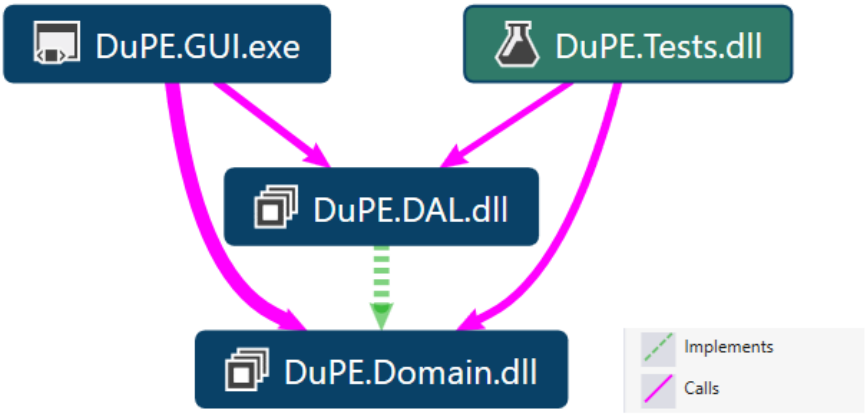
3.2 Funkce stávajícího řešení

- Vytvoření nového projektu DuPE (projekt DuPE obsahuje strukturu databáze, nastavení pro připojení k databázi a nastavená pravidla pro maskování)

- Uložení projektu DuPE do souboru
- Načtení projektu DuPE ze souboru
- Hromadné vytváření pravidel pro vybrané sloupce tabulek
- Automatické vytvoření pravidel pro všechny sloupce s podporovanými datovými typy
- Odebrání vybraných pravidel
- Odebrání všech pravidel
- Import a správu kolekcí dat
- Maskování databáze pomocí všech pravidel
- Maskování databáze pomocí všech nepoužitých pravidel
- Maskování databáze pomocí všech použitých pravidel

3.3 Struktura stávajícího řešení

Stávající řešení je postaveno na tří vrstvé architektuře, která zajistí oddělení jednotlivých částí aplikace jako GUI, aplikační logiku a práci s daty. Díky tří vrstvé architektuře lze jednoduše vyměnit nebo editovat jednotlivé vrstvy aniž by se kvůli úpravě jedné vrstvy musely upravovat vrstvy ostatní. Jednotlivé vrstvy jsou ve zvoleném IDE reprezentovány jako projekty s názvy Domain, DAL, GUI a navíc řešení obsahuje ještě jeden projekt určený pro testování s názvem Tests. Závislosti mezi projekty popisuje následující diagram.

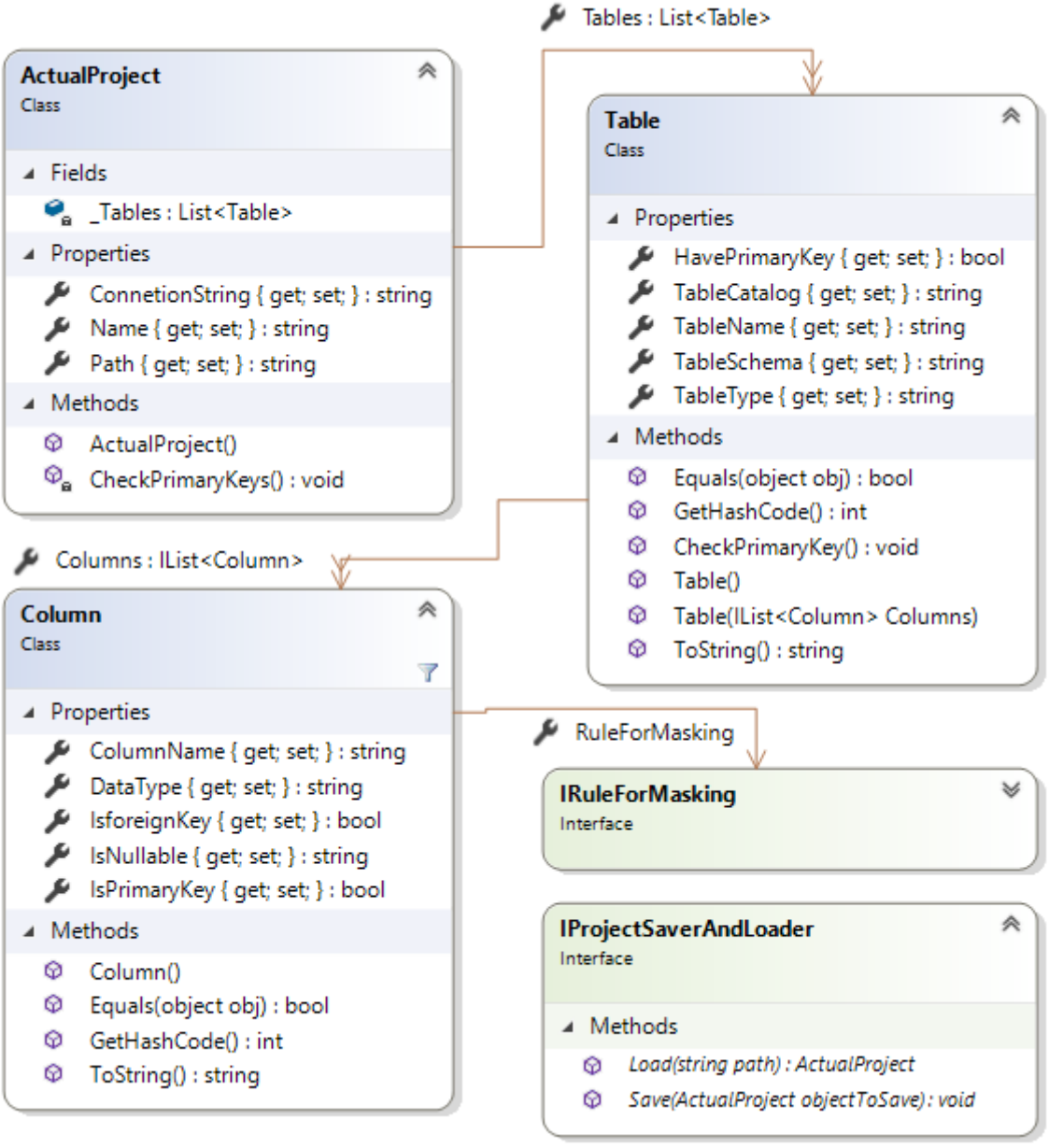


Obrázek 1: Diagram závisostí projektů stávajícho řešení

3.3.1 Domain - Doménová vrstva

Doménová vrstva nebo též aplikační vrstva obsahuje jádro aplikace, ve kterém je řešeno vytváření dat pro maskování hodnot a projekt DuPE, se kterým se v rámci celé aplikace pracuje.

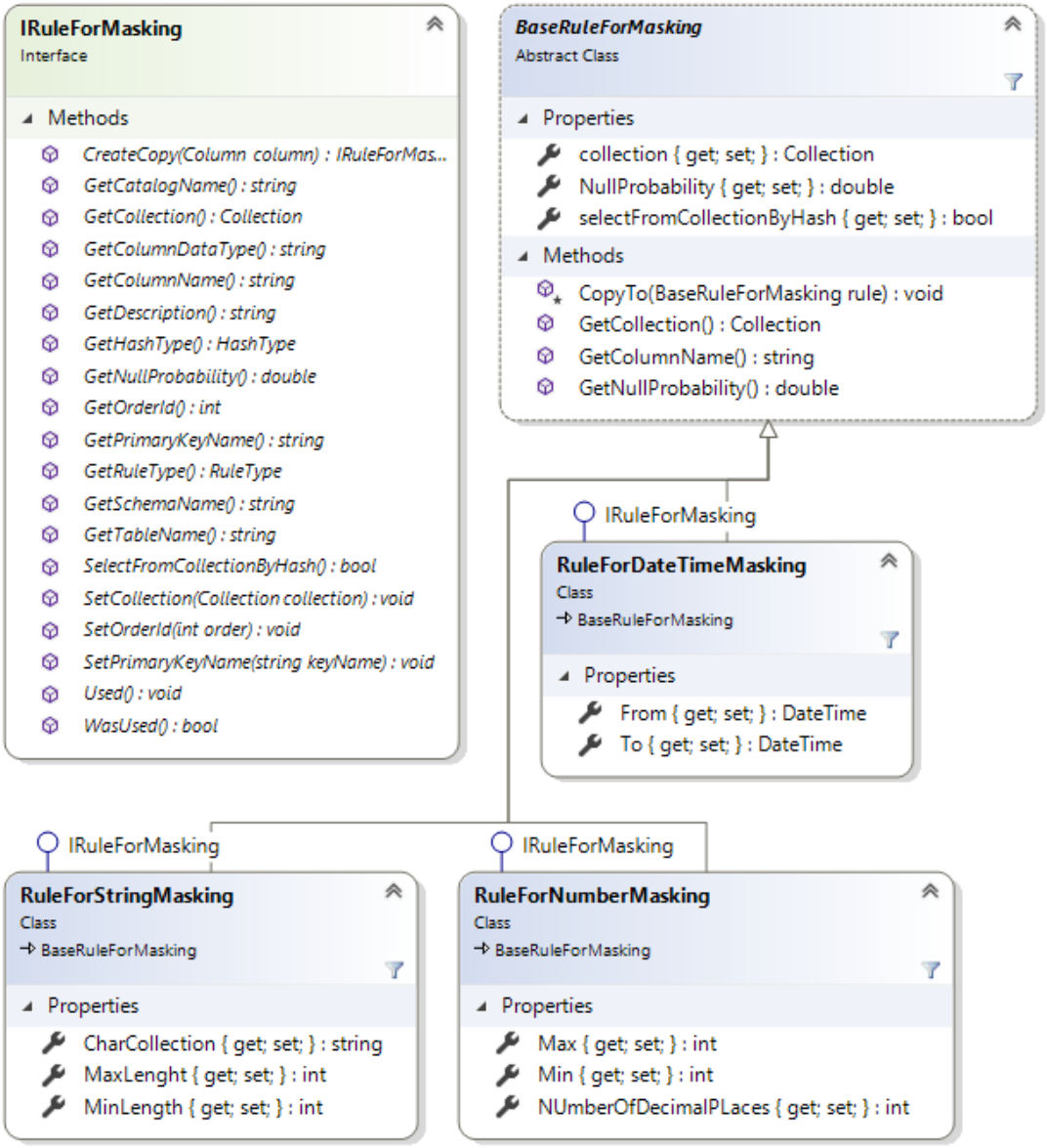
Projekt DuPE, obsahuje popis relační databáze, zejména popis tabulek a sloupců s integritními omezeními, potřebná nastavení k připojení k dané databázi a pravidla pro maskování. Pro možnost ukládání a načítání projektu pro jeho pozdější použití je vytvořeno rozhraní, které je implementováno v datové vrstvě.



Obrázek 2: Třídní diagram projektu DuPE

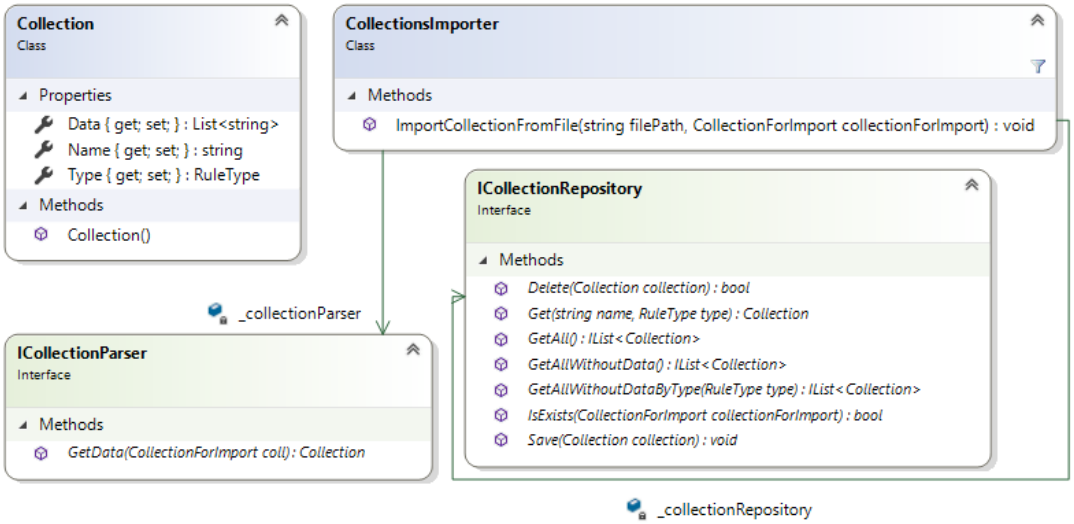
3.3.2 Pravidla pro maskování

Pro každý základní datový typ, který lze v aplikaci maskovat bylo vytvořeno pravidlo pro maskování, ve kterém lze nastavit jak způsob maskování, tak i zdroje nebo rozsahy hodnot, které se mají generovat.



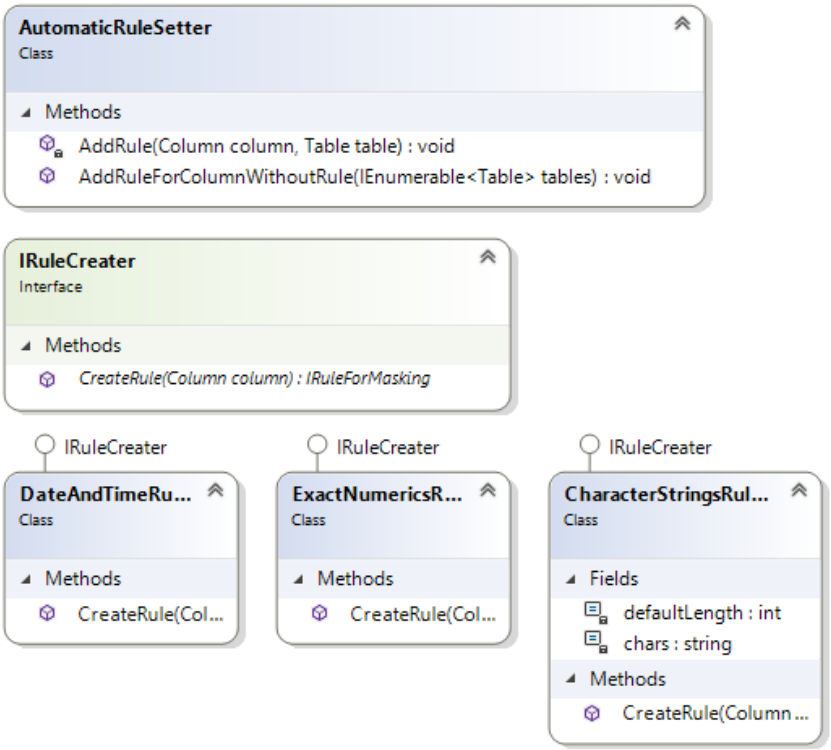
Obrázek 3: Třídní diagram pravidel pro maskování

Pravidla pro maskování obsahují také kolekce, ze kterých lze při maskování vybírat data, ať náhodně nebo deterministicky. Kolekce lze v aplikaci i spravovat, takže v doménové vrstvě najdeme i rozhraní pro jejich import a repositář.



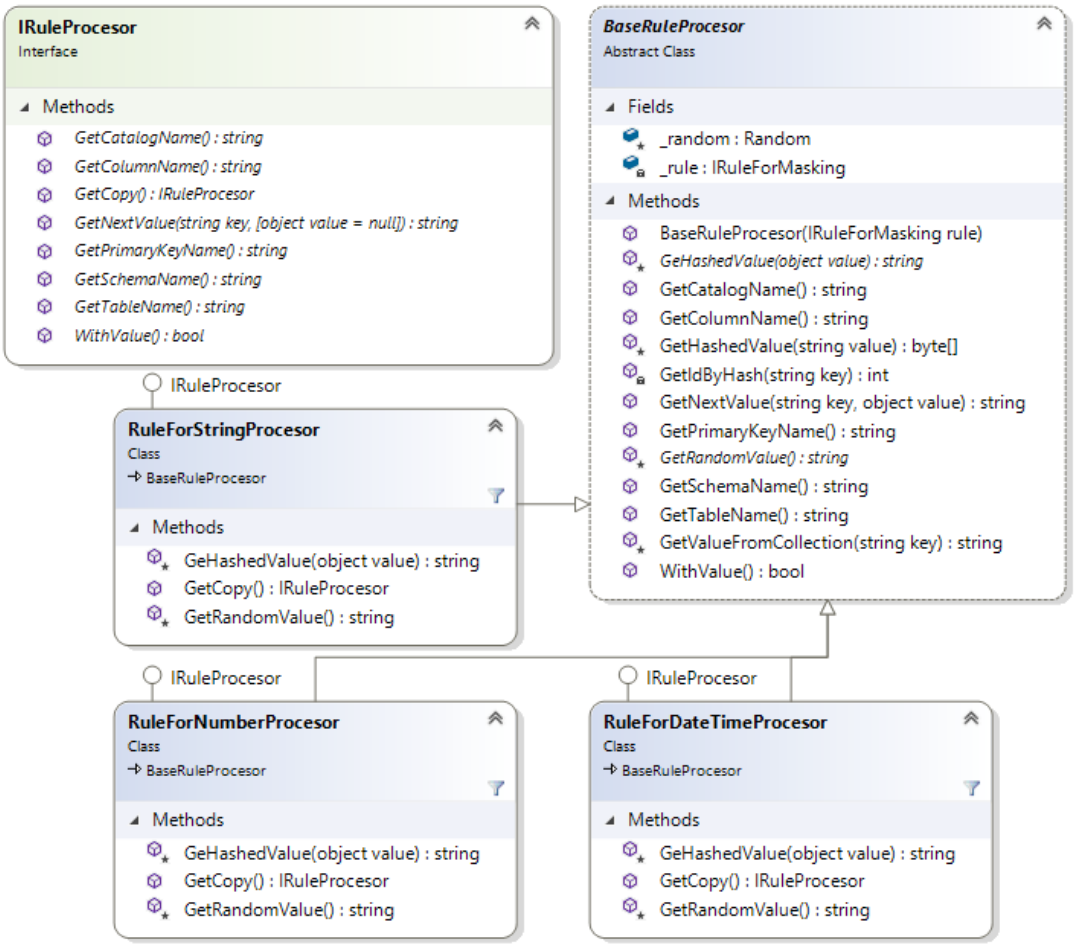
Obrázek 4: Třídní diagram datových kolkcí

V aplikaci lze automaticky vytvořit pravidla maskování pro všechny podporované základní typy a to pomocí implementace ve třídě AutomaticRuleSetter, která na základě datového typu vybere odpovídající RuleCreater a vytvoří pravidlo, které zohledňuje integritní omezení daného typu. Automaticky vytvořená pravidla jsou nastaveny vždy na náhodné generování hodnot.



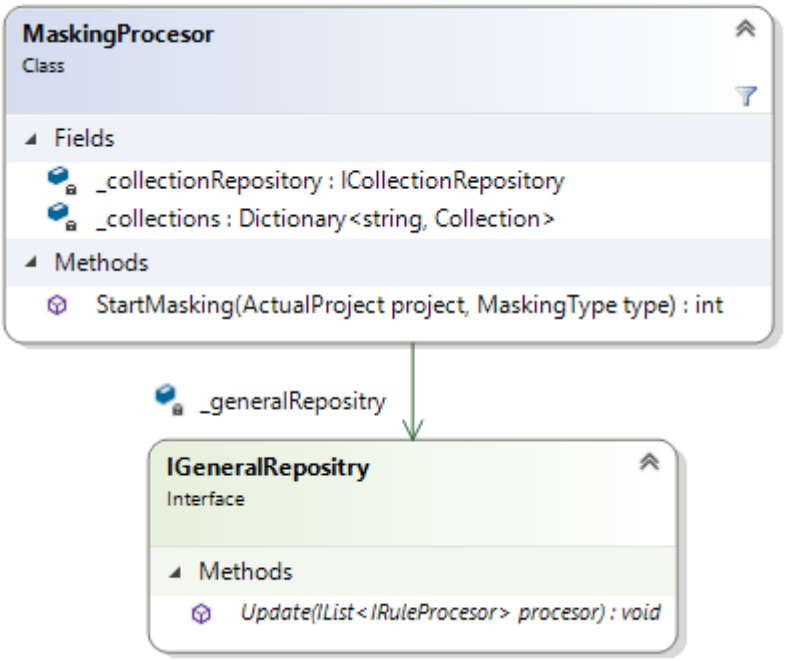
Obrázek 5: Třídní diagram AutomaticRuleSetter a RuleCreater

Generování hodnot z pravidel pro maskování řeší třídy, které implementují rozhraní IRuleProcessor, opět je pro každý základní podporovaný datový typ vytvořen jeden procesor, který podle zadaných kritérií vygeneruje novou hodnotu.



Obrázek 6: Třídní diagramy procesorů pravidel

Samotné maskování dat řeší třída MaskingProcesor. V první fázi vybere z projektu všechny pravidla pro maskování podle typu maskování a případně jim doplní data z kolekcí. Následně vytvoří instance tříd implementující rozhraní IRuleProcesor pro každé pravidlo zvlášť a předá je instanci rozhraní IGeneralRepository, která data uloží do databáze.

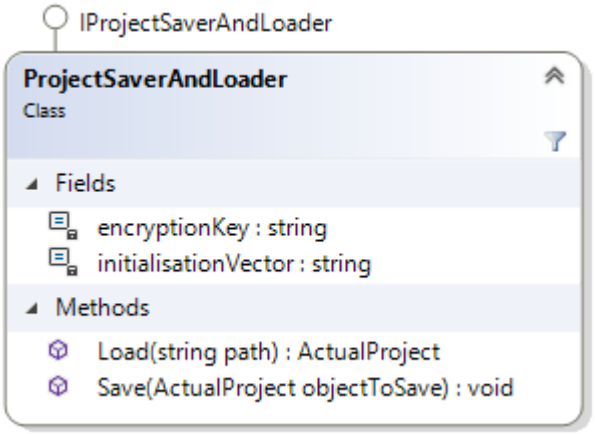


Obrázek 7: Třídní diagram procesoru maskování

3.3.3 DAL - Datová vrstva

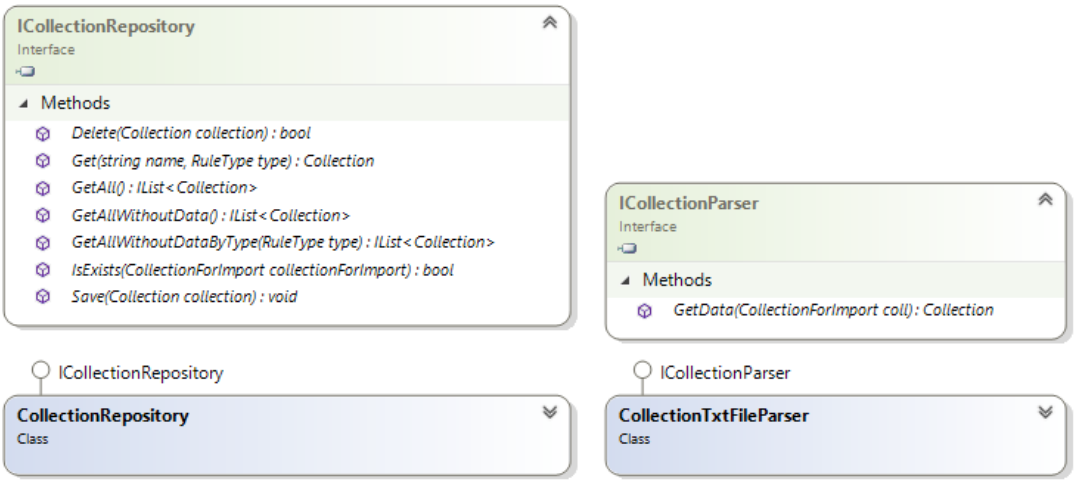
Datová vrstva řeší ukládání a načítání projektu DuPE, import, ukládání a načítání kolekcí a hlavně práci s databází.

Pro práci s projektem DuPE byla vytvořena třída ProjectSaverAndLoader jejíž implementace při ukládání převede projekt DuPE z doménové vrstvy do objektů vytvořených ze tříd popisující strukturu XML a tyto objekty serializuje do xml. A při načtení projektu naopak data deserializuje a převede do objektů doménové vrstvy.



Obrázek 8: Třídní diagram pro práci s projektem

Ukládání kolekcí se provádí obdobně, jako u projektu DuPE, také se serializuzuje do XML a při načtení opět deserializuje a to v implementaci třídy CollectionRepository. Pro import kolekcí byla vytvořena třída CollectionTxtFileParser, která řeší parsování dat kolekcí z textového souboru, kde na každém řádku je jedna hodnota. Parser obsahuje i validaci hodnot na předem zvolený typ dat, která by měl soubor obsahovat.



Obrázek 9: Třídní diagram pro práci s kolekcemi

Práci s databází lze rozdělit na dvě části: vyčtení struktury databáze a maskování dat.

Vyčtení struktury databáze

Struktura databáze se vyčte z pohledu systémového katalogu, přesněji z tabulek TABLES a COLUMNS pomocí knihovny nHibernate jedním příkazem, který vrátí všechny tabulky i s jejich sloupci. Poté se ještě doplní ke sloupcům primární a cizí klíče, které z databáze získáme použitím příkazů EXEC sp_pkeys a EXEC sp_fkeys pro každou tabulku.[8]

Maskování dat

Datová vrstva pro akci maskování dat obdrží od doménové vrstvy kolekci procesorů pravidel, kterou nejdříve rozdělí na dvě skupiny podle toho, zdali je k maskování potřeba hodnota maskovaného atributu.

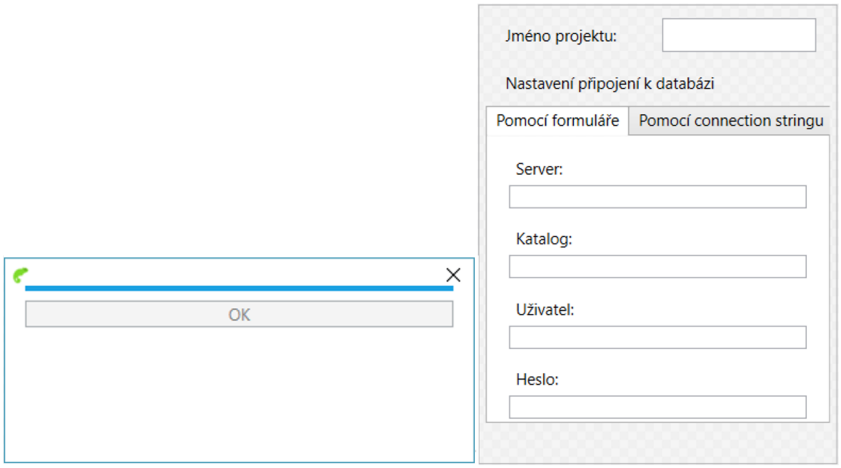
Poté se začne zpracovávat skupina procesorů, která potřebuje hodnotu atributu k maskování. Procesory se zpracovávají v cyklu každý zvlášť následujícím způsobem. Z tabulky, pro kterou je pravidlo v procesoru, se vyčte seznam záznamů s primárním klíčem a hodnotou, která má být maskována. Seznam záznamů se pak prochází a při každém průchodu procesor vygeneruje novou hodnotu, kterou aktualizuje původní pomocí příkazu update a podmínky na primární klíč.

Druhá skupina, která nepotřebuje hodnotu atributu k maskování, se nejdříve rozdělí do dalších podskupin podle tabulky, nad kterou jsou použity pravidla maskování. Tyto podskupiny

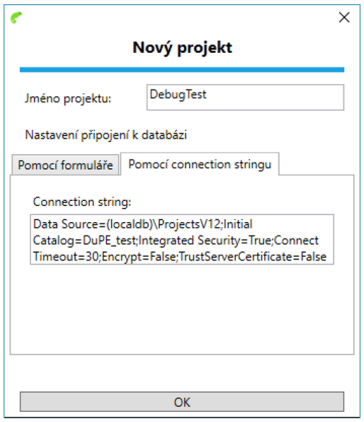
se poté zpracují naráz. Pro celou podskupinu se vyčte seznam primárních klíčů a vytvoří se příkaz update, kterým lze aktualizovat všechny maskované atributy dané tabulky. Seznam záznamů se pak prochází a pro každý záznam se vytvořený příkaz update doplní o vygenerované hodnoty z procesorů a primární klíč záznamu, poté se příkaz provede. Při větším počtu záznamů jak 5000 jsou vytvořena čtyři vlákna, aby se maskování mohlo provádět paralelně. Při počtu menším než 5000 záznamů se maskování provádí pouze na jednom vlákně

3.3.4 GUI - Prezentační vrtva

Prezentační vrstva řeší získávání vstupů od uživatele, zobrazení stavu projektu a spouštění akcí nad projektem. Většina GUI je vytvořena z tzv. userControls (až na MainWindow), ty jsou potom vsazeny do obyčejného nebo dialogového okna. Díky tomu lze jednoduše měnit vzhled celého projektu a znovu použít již vytvořené prvky.

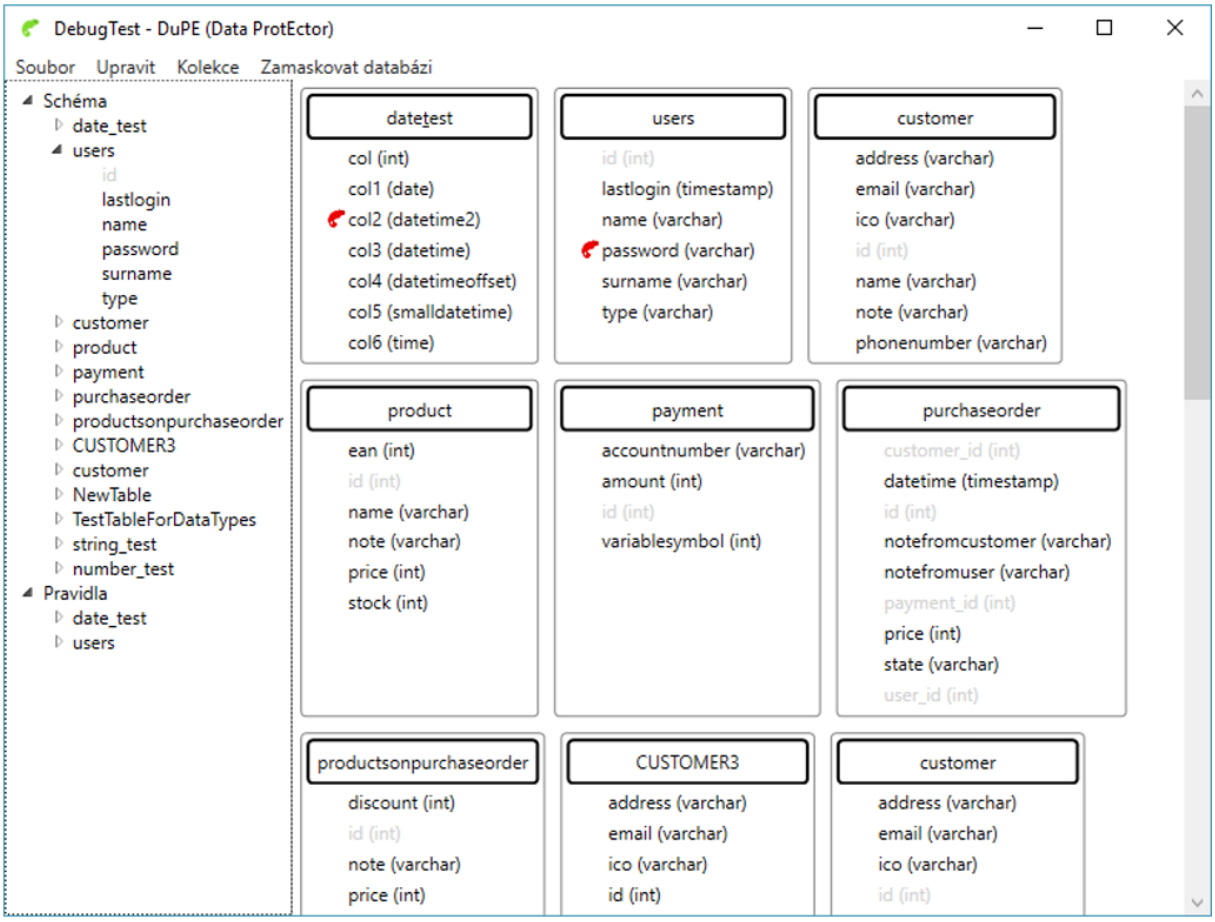


Obrázek 10: Ukázka UcerControlu a okna



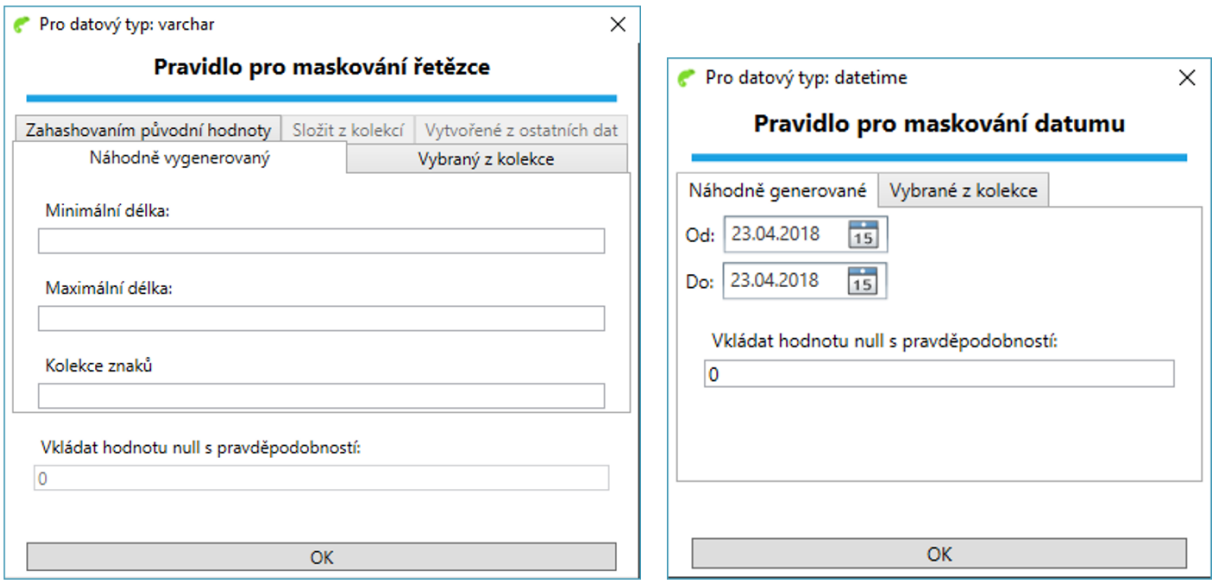
Obrázek 11: Ukázka UserControlu vloženého do okna

Základem GUI je hlavní okno MainWindow sloužící pro zobrazení projektu a práce s ním.

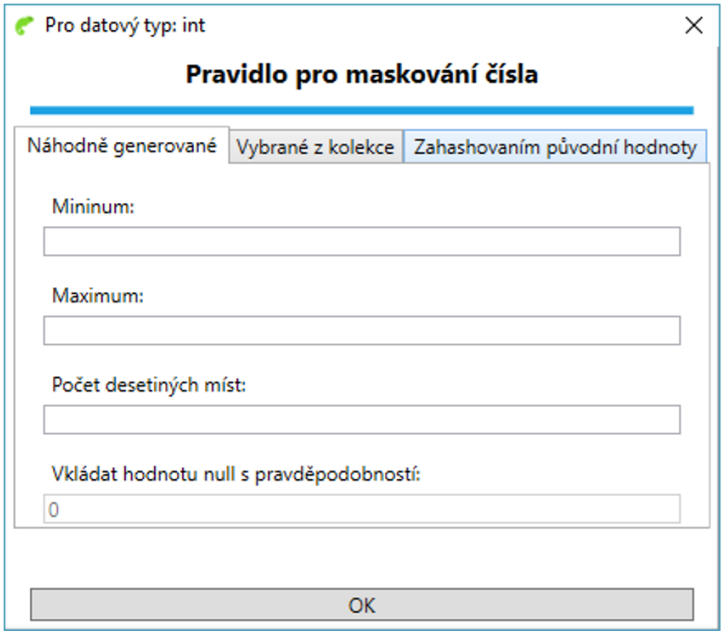


Obrázek 12: Hlavní okno aplikace MainWindow

Pro vytváření pravidel maskování jsou vytvořeny pro každý základní datový typ zvlášť user-Controly.

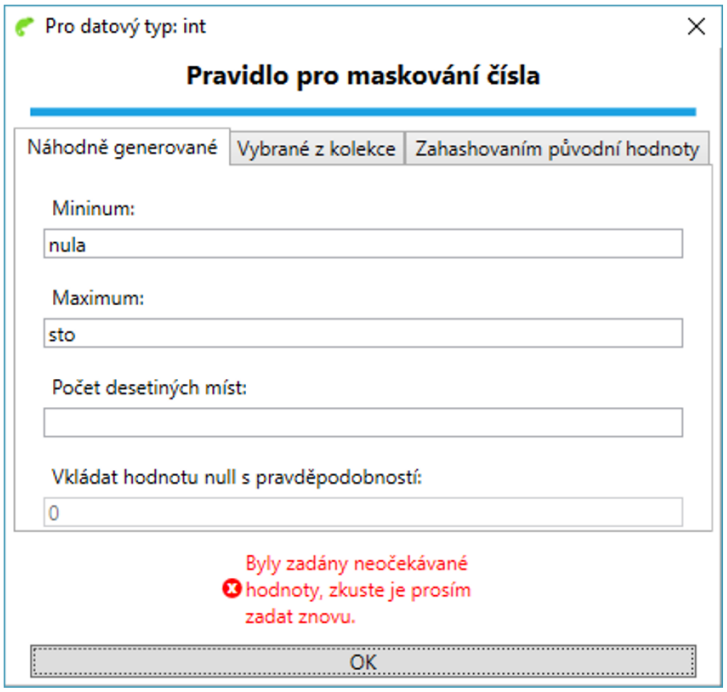


Obrázek 13: UserControly pro maskování řetězce a datumu



Obrázek 14: UserControl pro maskování čísla

Vrstva řeší i validitu vstupů od uživatele a v případě, že vstup není ve správném formátu, zobrazí uživateli upozornění s chybovým hlášením.



Obrázek 15: Ukázka upozornění na chybně zadané hodnoty

Touto vrstvou se spouští a ukončuje celá aplikace, a proto do ní bylo implementováno logování spouštění, ukončení a pádu aplikace pomocí knihovny log4net.

3.3.5 Tests - komponenta pro testy

Poslední projekt (projekt v IDE) s názvem Tests byl vytvořen pro vytváření jednotkových testů a obsahuje testování dvou základní funkcí aplikace a to práci s kolekcemi a funkce procesoru pravidel.

4 Doplněné technologie

Do aplikace byly doplněny technologie, které zjednodušily její vývoj a pomohly aplikaci vytvořit agilnější.

Ninject

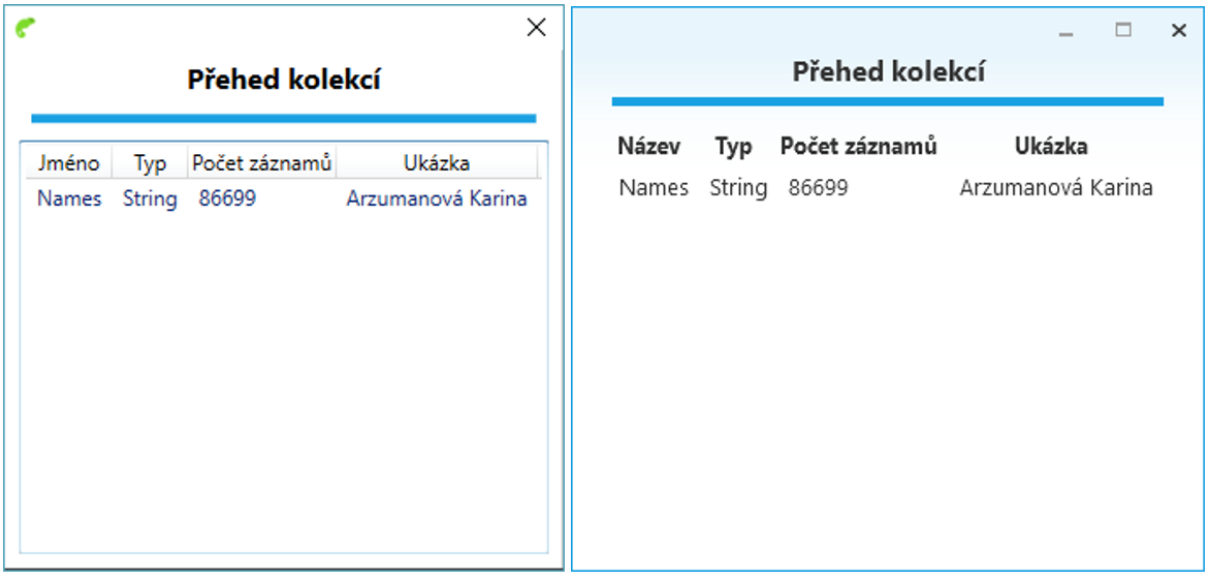
Ninject je framework, který implementuje injektor závislostí pro aplikace .NET. Pomocí Ninjectu lze v aplikaci snadno použit návrhový vzor DI, díky kterému bude aplikace lépe testovatelná a univerzálnější.[9]

Moq

Je technologie určená k Mockování neboli k náhradě reálné funkcionality zjednodušeným mechanismem. Díky této funkcionalitě se zjednoduší testování, například pokud k testu potřebujeme instanci repozitáře popsaného rozhraním, tak Moq ji podle daného rozhraní vytvoří a dovolí nám nastavit implementaci zvolených metod, ve kterých si například můžeme jednoduše vrátit předpřipravený objekt.[10]

ModernUI.WPF

ModernUI. WPF je sada ovládacích prvků a stylů pro aplikace WPF. Díky této technologii aplikace získá modernější a líbivější vzhled.[11]



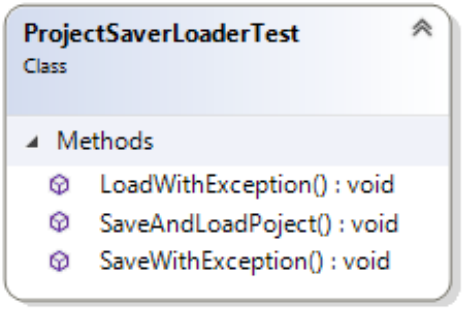
Obrázek 16: Ukázka změny vzhledu při použití knihovny ModernUI

5 Vytvoření unit testů a refactoring

Stávající kód byl doplněn o několik sad jednotkových testů. Všechny tyto sady testují základní funkčnost testované komponenty včetně zadání špatných vstupních parametrů, kde je v průběhu testu očekávána výjimka, pokud v takovém případě výjimka nastane, test je označen jako úspěšný, když nenastane, test je označen jako neúspěšný. V testech, kde to má smysl, je návratová hodnota metody validována pomocí třídy Assert, která v případě, že je očekávaný výsledek jiný než výstup z testované metody vyvolá výjimku a test skončí jako neúspěšný. Při testech je použita i inicializace testovací třídy pro přípravu testovacího prostředí (například pro nachystání potřebných souborů na souborovém systému).

5.1 Testování práce s projektem

V sadě testů pro práci s projektem je hlavně testováno správné uložení a opětovné načtení projektu tak, aby všechny informace zůstaly zachovány. Při testování se v prvním kroku uloží předpřipravený projekt na disk a poté se vyčte. V dalším kroku se předpřipravený projekt porovná s vyčteným, a pokud nedojde k nalezení rozdílu je test úspěšný. Další testy v této sadě testují chování repozitáře při práci s neexistujícím souborem nebo umístěním.



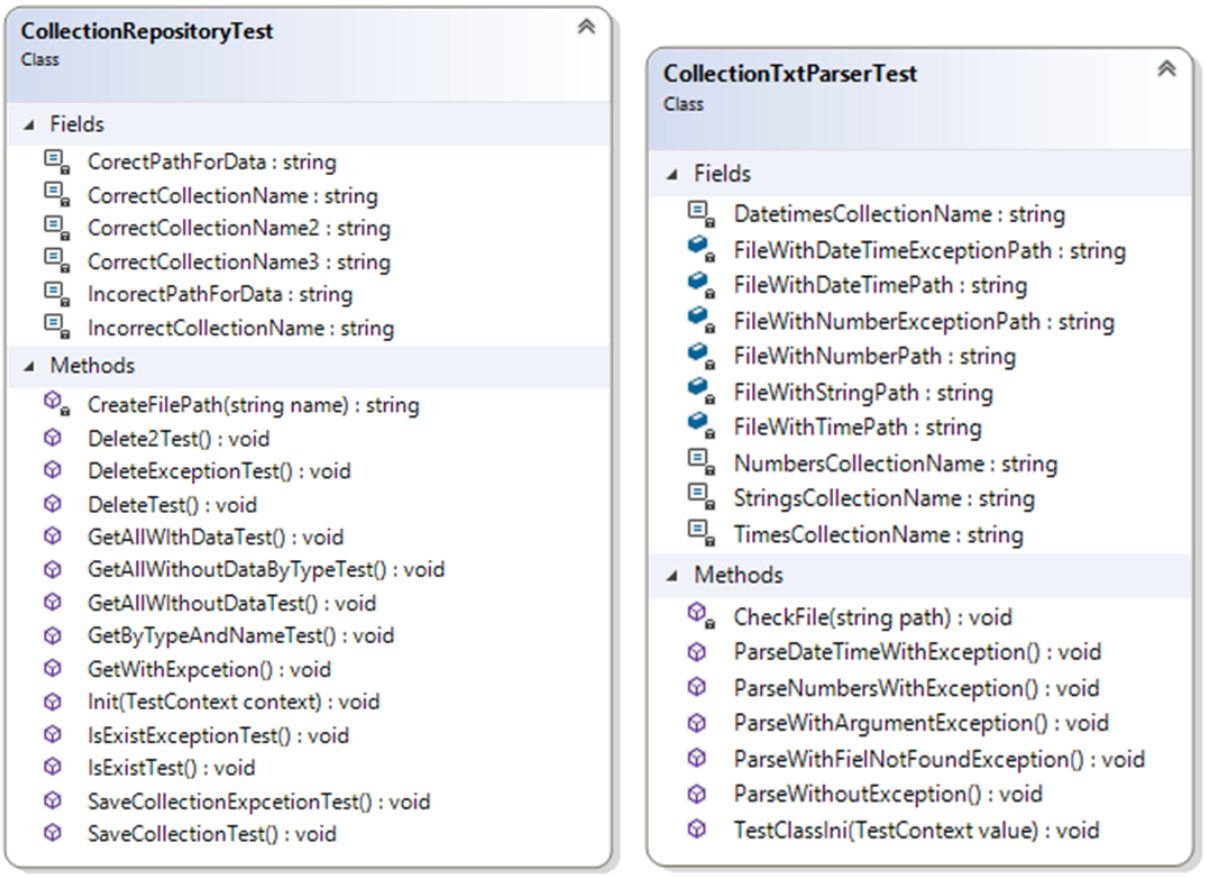
Obrázek 17: Třídní diagram pro testování práce s projektem DuPE

5.2 Testování práce s kolekcemi

Stávající řešení sice již obsahuje testování práce s kolekcemi, ale jsou zde naimplementovány pouze tři testy, kde se nevaliduje výsledek. Takže úspěšný test znamená, že kód neskončil výjimkou. Tyto testy byly přepracovány a doplněny o řadu dalších. Testování práce s kolekcemi je rozděleno do dvou sad testů. Testování repozitáře kolekcí a testování parsování dat pro kolekce.

Pro testování repozitáře kolekcí je před spuštěním sady testů provedena kontrola a případně příprava testovacího prostředí. Testovacím prostředí se v tomto konkrétním případě myslí souborový systém, který obsahuje potřebné soubory pro testy. Díky předpřipravenému prostředí víme jaké hodnoty očekávat na výstupu a ty lze potom validovat.

Další sada testů, která byla přidána, je testování parsování dat pro kolekce. I v této sadě testů se před zahájením testování spustí kontrola a příprava testovacího prostředí, aby bylo možné výstupy metod validovat. Testy se zaměřují hlavně na správné parsování hodnot.

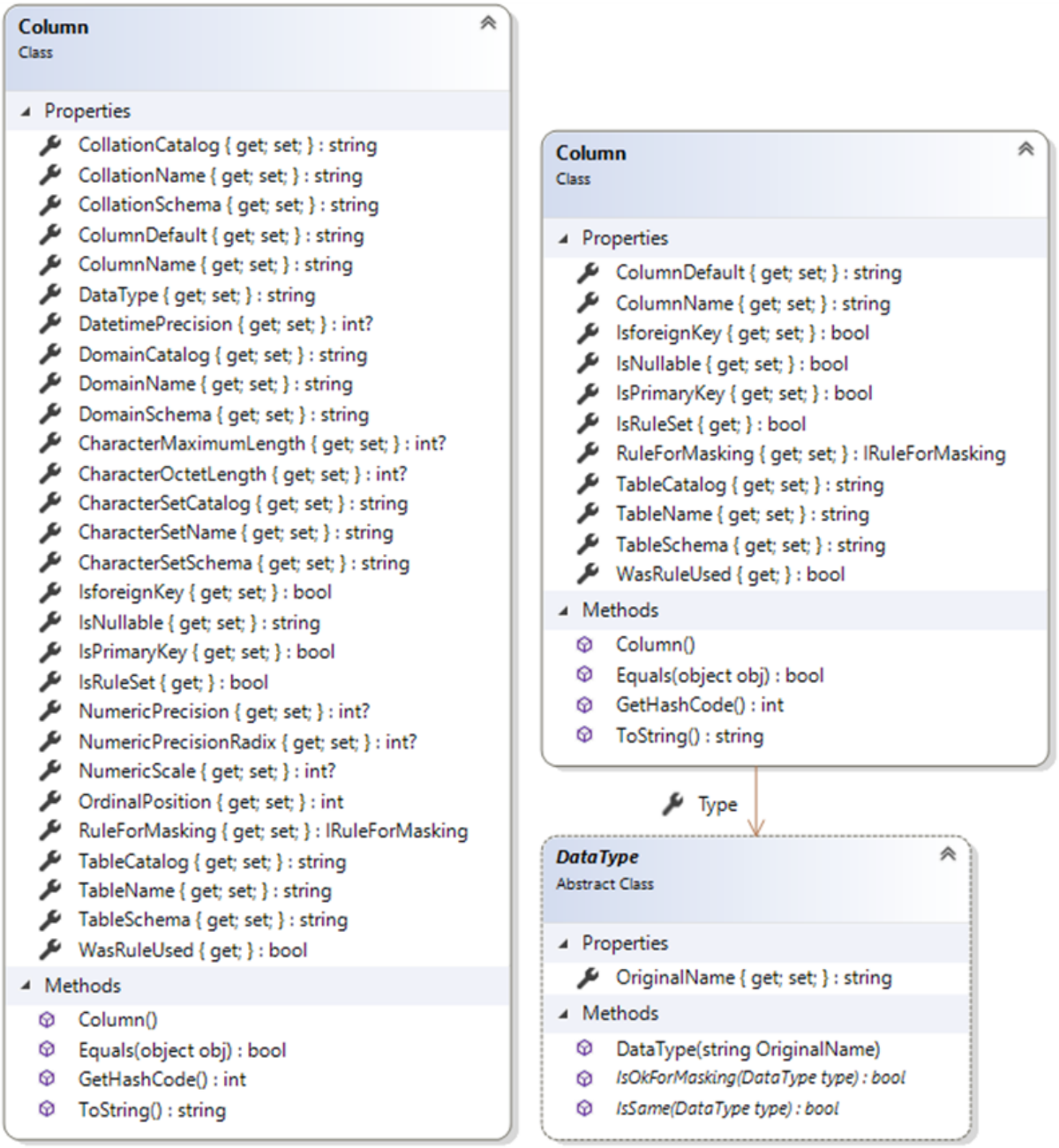


Obrázek 18: Třídní diagramy pro testování práce kolekcí

5.3 Refactoring

Stávající podoba aplikace byla navrhnuta pro práci s jednou konkrétní databází a to se i přes použití třívrstvé architektury projevilo na doménové vrstvě. Například třída Column obsahuje atributy, které jsou převzaty z tabulky column informačního schéma sql server databáze. První fáze refactoringu tedy představovala kontrolu všech tříd, zdali neobsahují zbytečné atributy nebo zdali nejsou málo obecné.

V již zmíněná třídě Column byly odstraněny nadbytečné atributy a pro atributy popisující datový typ byly vytvořeny třídy nové.



Obrázek 19: Porovnání třídních diagramů pro Column před a po refactoringu

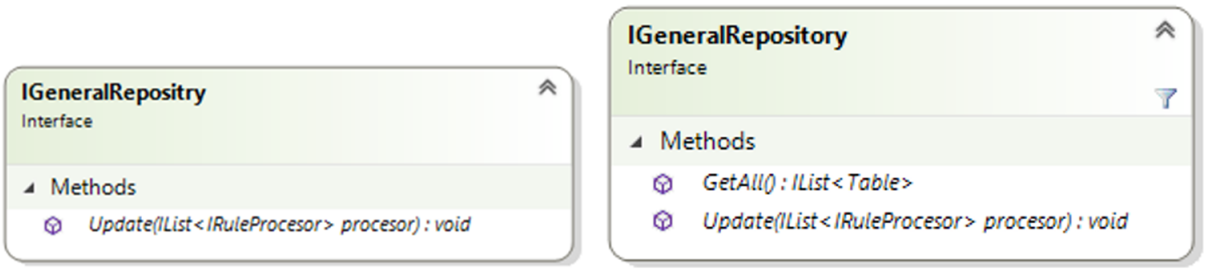
Obrázek 20: Třídní diagram nově vytvořeného datového typu

Třídy implementující rozhraní IRuleProcesor používaly k nastavení vybraných parametrů rozhodování pomocí konstrukce switch, kde vstupem byl textový řetězec. Textový řetězec představoval datový typ převzatý z SQL Serveru. Tato implementace byla odstraněna a po refactoringu se potřebné informace berou z pravidla pro maskování. Stejný postup nastavení vybraných parametrů byl implementován i ve třídách implementujících rozhraní IRuleCreater, nová implementace byla vyřešena stejně jako v předešlém případu.

V druhé fázi refactoringu bylo cílem odhalit kód nebo použité techniky, které by mohly negativně ovlivnit další vývoj aplikace.

V aplikaci se s databází pracuje při dvou úlohách. První je vyčtení struktury a druhou je maskování dat. Pro třídu, která implementuje maskování dat je definováno rozhraní IGeneralRepository díky čemuž lze snadno nahradit stávající implementaci. To samé ovšem už neplatí o vyčtení struktury databáze, která nemá nadefinované rozhraní, takže dochází k přenesení závislostí. Díky tomuto přenesení závislostí by bylo obtížnější aplikaci dále rozšiřovat. Proto bylo

rozhraní IGeneralRepository rozšířeno o metodu, která vyčte strukturu databáze.



Obrázek 21: Porovnání třídních diagramů pro IGeneralRepository před a po refactoringu

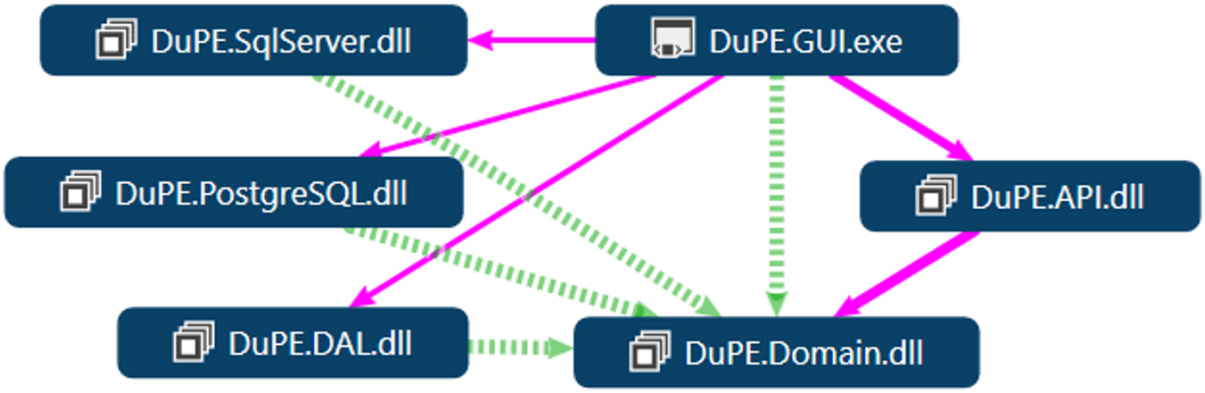
Jako další slabý článek pro možný rozvoj aplikace byl nalezen způsob práce s texty určenými pro uživatele. Tyto texty byly uloženy na různých místech v kódu jako textové řetězce, tento způsob by znesnadnil možný překlad aplikace do dalšího jazyka nebo jen kontrolu textů. Texty nebyly často ani uloženy v proměnných, které by popisovaly význam textů. Tento problém byl odstraněn použitím souborů Resources.resx, do kterých se texty ukládají a v kódu se na ně poté jen odkazuje. Použití souborů Resources.resx umožnuje i lokalizaci aplikace, kde pro přidání dalšího jazyka stačí přidat soubor resx se stejným názvem doplněným o lokalizaci. Například když budeme mít uloženy anglické texty v souboru StringsResource.resx (bez lokalizace bude tento soubor použit jako defaultní) tak pro českou verzi vytvoříme soubor s názvem StringsResource.cs-CZ.resx.

Pro práci s takto uloženými texty je pak vhodné použít nástroj ResXManager, který se doinstaluje do Visual Studia jako doplněk. Tento nástroj dokáže zobrazit všechny překlady jednoho textu na jednom řádku, díky čemuž jsou snadno dohledatelné chybějící překlady. Dále umožňuje texty exportovat a importovat, což ulehčí práci programátora a mnoho dalšího.[12]

6 Rozšíření aplikace

Na základě nových požadavků na aplikaci byla navržena nová struktura aplikace. Nová struktura vychází z původního návrhu, který je doplněn o projekty SqlServer(pro práci s SQL serverem), PostrgeSQL (pro práci s PostgreSQL) a API (pro zpřístupnění základních funkcionalit aplikace). Projekt DAL se tedy rozdělil na dva. Na projekt s názvem DAL, kde zůstala práce s kolekcemi a projektem DuPE. A na již zmíněný projekt SqlServer.

Tímto rozdělením bylo dosaženo přehlednější struktury a vetší škálovatelnosti celé aplikace.



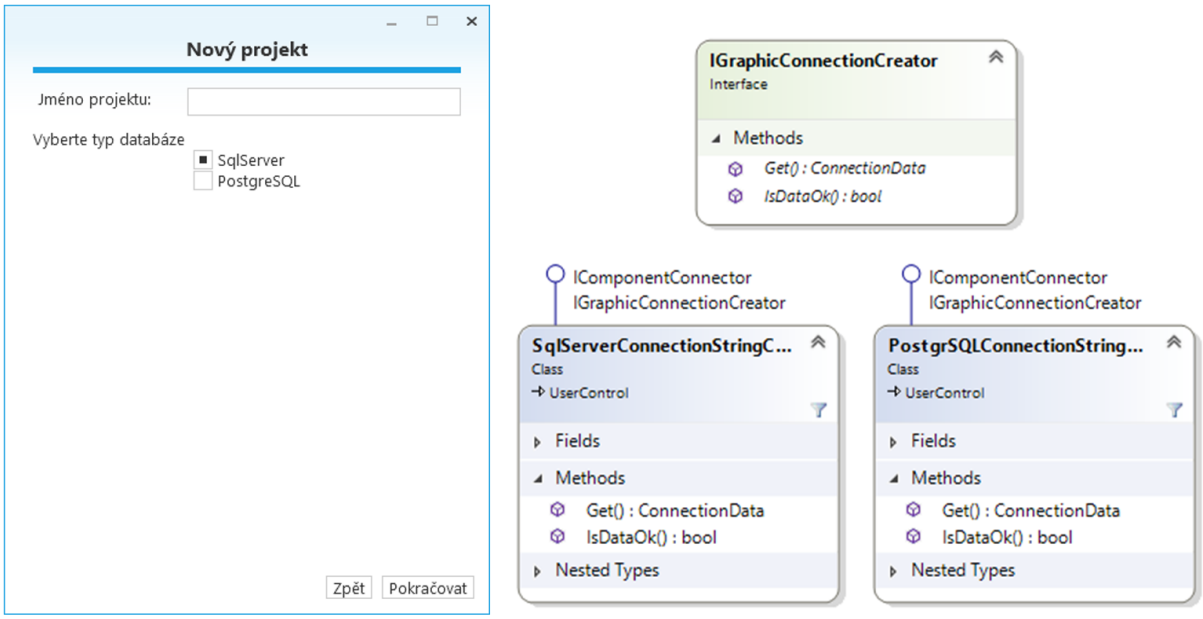
Obrázek 22: Diagram závislostí projektů nové struktury aplikace

6.1 Podpora PostgreSQL databáze

Aby aplikace umožnovala podporu více databází a zachovala se škálovatelnost aplikace bylo potřeba upravit grafické rozhraní pro uživatele, doménovou vrstvu a přidat implementaci práce s PostgreSQL databází do příslušného projektu.

6.1.1 Změna GUI

Stávající implementace okna pro vytvoření nového projektu byla navrhnuta pouze pro jeden typ databáze. Nová implementace byla navrhnuta tak, že pro přidání další databáze stačí vytvořit UserControl s potřebnými formuláři pro nastavení připojení k databázi a implementující rozhraní IGraphicConnectionCreator. Takto vytvořené userControly se vloží jako slovník (kde klíč je název databáze) do konstruktoru okna nového projektu. Okno nového projektu zobrazí uživateli seznam databází (neboli klíčů), uživatel vybere jednu z možností a okno zobrazí UserControl podle výběru. Potom, co uživatel vyplní formulář pro připojení k databázi se pomocí metod nadefinovaných rozhraním vyčtou data o databázi.



Obrázek 23: Ukázka nového GUI pro nový projekt spolu s třídním diagramem

6.1.2 Změna doménové vrstvy

Doménová vrstva obsahovala rozhraní pro vytvoření connection stringu a i její implementaci pro SQL Server. Tato implementace byla přemístěna do příslušného projektu a taky byla vytvořena nová implementace pro PotgreSQL. Jelikož se třída pro vytvoření connection stringu přesunula z okna nového projektu do jednotlivých usercontrolů databází, zanikla potřeba rozhraní. Takže rozhraní bylo odstraněno.

6.1.3 Implementace PostgreSQL

Začlenění další databáze do datové vrstvy, která již obsahuje dva projekty DAL a SqlServer, je díky vytvořenému rozhraní IGeneralRepositry z hlediska návrhu jednoduché. Stačí vytvořit třídu, která zmíněné rozhraní implementuje. Díky dodržení rozhraní bude možné danou implementaci použít i v ostatních vrstvách bez nutnosti dalších úprav.

Implementace, která se vloží do nového projektu s názvem PostgreSQL, je rozdělena na dvě části. První se zabývá vyčtením struktury databáze a druhá řeší maskovaní dat ve smyslu uložení dat do databáze.

Vyčtení struktury databáze

Pro práci s databází PostgresSQL, stejně jako pro práci s SQL Serverem, byl použit nHibernate. Aby bylo možné z databáze vyčítat data pomocí nHibernate, musely se nejdříve vytvořit třídy, které budou reflektovat entity relační databáze. Třídy byly vytvořeny podle tabulek tables a

columns z pohledu systémového katalogu. Z vytvořených tříd se vytvořily třídy pro mapování na příslušné tabulky z databáze.

Vytvořené mapování se vloží spolu s connection stringem do nastavení configurace připojení k databázi, které vytvoří SessionFactory. Ze SessionFactory poté při dotazech otevíráme relaci spojení (session), pomocí které se dotazujeme databáze. Po těchto krocích jsme již schopni jednoduše vyčíst základní strukturu databáze.

Po vyčtení základní struktury databáze je třeba označit u sloupců, zdali se nejedná o primární nebo cizí klíče. Informace o primárních a cizích klíčích se získá z pohledu systémového katalogu konkrétně z tabulek key_column_usage a table_constraints následujícím dotazem, který spustíme pro každou tabulku.

Získaná data je třeba ještě převést do objektů doménové vrstvy. Pro tento účel byla vytvořena třída Converter, která všechny tabulky a sloupce převede do doménových objektů a sloupcům vytvoří datové typy podle jejich vlastností.

Maskování dat

Maskování dat nebo přesněji uložení maskovaných dat do databáze je část aplikace, která nejvíce ovlivní rychlost celého procesu maskování. Proto návrhu a implementaci této části bylo věnováno nejvíce času. Nejdříve bylo potřeba zvážit klady a zápory různých možností, jak data uložit do databáze.

Následuje výčet možností ukládání dat s jejich popisem a hodnocením.

- Jednoduchý update pro každý záznam

  Ukládání dat pomocí příkazu SQL update pro každý záznam s podmínkou na shodu primárního klíče. Tato možnost byla použita v původní aplikaci a její dobré výsledky spočívaly v testování na lokální databázi při použití SSD, tuto skutečnost odhalila tato práce v části testování výkonu. Implementačně je tato metoda jednoduchá, ale zato výkonnostně nedostačující.

- Bulk insert do cílové tabulky

  Pro ukládání dat pomocí hromadného vkládání do cílové tabulky, by se nejdříve musely vyčíst všechny data z tabulky (i atributy které k maskování nebude potřeba). V případě, že primární klíč tabulky bude jinde použit jako cizí klíč, musela by se vypnout kontrola na integritní omezení a to buď jen na dotčených tabulkách, nebo nad celou databází. Poté by se musely záznamy z tabulky odstranit a to lze provést dvěma způsoby.

  První je vytvoření kopie tabulky a smazání původní tabulky pomocí sql příkazu drop a následným přejmenováním kopie podle původní tabulky. Bylo by to výkonově velice výhodné, ale muselo by se zajistit, že nově vytvořená tabulka má všechny vlastnosti jako původní (např. nastavení indexů), což by bylo implementačně složité.

Druhá možnost je použít SQL příkaz delete bez podmínky, což je výkonově horší, ale odpadají zvýšené nároky na implementaci.

Po odstranění záznamů by se zamaskovaná data nahrála do databáze pomocí hromadné operace bulk insert a případně by se opět aktivovala kontrola na integritní omezení.

Toto řešení by bylo výkonnostně lepší než předešlé, ale aktivace a deaktivace integritních omezení není korektní způsob, jak pracovat s databází. Ani vyčítání dat, které nejsou potřeba, není efektivní způsob práce.

- Bulkinsert do nové tabulky a update

  V prvním kroku by se vyčetly data z cílové databáze (ale jen potřebná k maskování). Následně by se vytvořila dočasná tabulka, do které se pomocí bulkinsertu vloží zamaskovaná data a provedl by se update cílové tabulky pomocí dat z dočasné tabulky.

  A nakonec se odstraní dočasná tabulka pomocí sql příkazu drop.

  Toto řešení je z hlediska práce s databází korektní, nenačítá zbytečná data a obsahuje hromadné operace, takže jeho výkon by už mohl být zajímavý.

- Bulkinsert do nové tabulky a upsert

  V prvním kroku by se vyčetly data z cílové databáze (ale jen potřebná k maskování). Následně by se vytvořila dočasná tabulka, do které se pomocí bulkinsertu vloží zamaskovaná data a provedl by se upsert (neboli příkaz insert s klauzulí on conflict) nad cílovou tabulkou pomocí dočasné tabulky.

  A nakonec se odstraní dočasná tabulka pomocí SQL příkazu drop.

  Toto řešení stejně jako předešlé nenačítá zbytečná data a obsahuje hromadné operace.

Z výše popsaných možností byly první dvě zavrhnuty. První kvůli slabému výkonu a druhá kvůli ne příliš korektní práci s databází. Ze zbylých dvou možností nebylo možné vybrat bez testování jejich výkonu, takže se provedly testy.

Testování Obě řešení jsou totožné až na aktualizaci cílové tabulky, takže testování se zaměřilo pouze na SQL příkaz update a upsert. Pro testování byly vytvořeny dvě tabulky Customer a MaskedCustomer. Do obou tabulek byly vloženy 3 miliony náhodně vygenerovaných záznamů.

Lineární zápis vytvořených tabulek

Legenda: **Tabulka**, primární klíč, *cizí klíč*, atribut

- **Customer**(ID, Name, Address, PhoneNumber, Email, ICO, Note)
- **MaskedCustomer**(ID, Name, Address, PhoneNumber, Email, ICO, Note)

Datový model vytvořených tabulek

Tabulka 1: Tabulka Customer (Zákazník)

	Datový typ	Klíč	Null	Index	Význam
ID	Int	Primární	N	A	Identifikátor
Name	Varchar(100)		N		Celé jméno
Address	Varchar(200)		N		Adresa
PhoneNumber	Varchar(20)				Telefonní číslo
Email	Varchar(254)				Email
ICO	Varchar(20)				Identifikační číslo osoby
Note	Varchar(200)				Poznámka

Tabulka 2: Tabulka MaskedCustomer (Maskovaný Zákazník)

	Datový typ	Klíč	Null	Index	Význam
ID	Int		N		Identifikátor
Name	Varchar(100)		N		Celé jméno
Address	Varchar(200)		N		Adresa
PhoneNumber	Varchar(20)				Telefonní číslo
Email	Varchar(254)				Email
ICO	Varchar(20)				Identifikační číslo osoby
Note	Varchar(200)				Poznámka

Tabulka 3: Výkon vkládní záznamů do PostgreSQL zobrazený pomocí tabulky

	Počet záznamů	Počet stránek	Čas vkládání[s]
Customer	3 000 000	47799	102
MaskedCustomer	3 000 000	47808	104

Tabulka 4: Velikosti vytvořených tabulek

Název tabulky	Velikost tabulky	Velikost indexů	Celková velikost
Customer	374 MB	64 MB	438 MB
MaskedCustomer	374 MB	0 MB	374 MB

Testované operace byly pro účely testování označeny jako S1 pro upsert a S2 pro update. Obě operace prováděly stejnou úlohu a to aktualizaci dvou atributů tabulky Customer pomocí dat z tabulky MaskedCustomer.

```
INSERT INTO Customer (id,name,address)
select id,name,address from MaskedCustomer
ON CONFLICT(id) DO UPDATE SET name = EXCLUDED.name,address = EXCLUDED.address
```

Výpis 1: Operace S1 upsert

```
UPDATE Customer
SET name = MaskedCustomer.name, address = MaskedCustomer.address
FROM MaskedCustomer
WHERE Customer.id = MaskedCustomer.id;
```

Výpis 2: Operace S2 update

Samotné testování spočívalo ve spuštění jednotlivých operací s výpisem plánu vykonání dotazu, ze kterého byly vyčteny a zapsány do tabulky hodnoty pro zhodnocení výsledků.

Tabulka 5: Přehled výkonu operací S1 a S2 v prvním testu

	shared hit block	shared readblock	LA	total cost	time[ms]
S1	33071751	168045	33239796	77797	90159
S2	26958943	215379	27174322	963938	108744

Výsledky testů ukazují, že operace S1 má o 22,3% větší počet logických přístupů na disk, ale zase naopak její celková cena je o 1239% menší. Po úvaze zdali by nešlo docílit ještě lepšího výsledku, byl přidán do tabulky MaskedCusmer Index pro atribut ID a test se zopakoval.

Tabulka 6: Velikost tabulek po vytvoření indexu pro atribut ID

Název tabulky	Velikost tabulky	Velikost indexů	Celková velikost
Customer	374 MB	64 MB	438 MB
MaskedCustomer	374 MB	64 MB	438 MB

Tabulka 7: Přehled výkonu operací S1 a S2 v druhém testu po přidání indexu

	shared hit block	shared readblock	LA	total cost	time
S1	35798283	215896	36014179	77799	89030
S2	26750811	218273	26969084	467336	93709

Výsledky testů po přidání indexu ukazují, že operace S1 má o 33,6% větší počet logických přístupů na disk, ale zase naopak její celková cena je o 600% menší.

Během testování se PostgreSQL databáze dostala do stavu, kdy se chovala jako by u tabulek neexistoval index a vždy prováděla sekvenční scan, v takovém případě běh operace S2 trval až 20 minut ale běh operace S1 trval stále kolem minuty a půl. Přinutit databázi využít indexy se podařilo až po použití příkazu VACUUM ANALYZE, který aktualizuje statistiky používané pro plánování dotazů.[13]

Z výsledků obou testů vyplývá, že operace S1 je náročnější na počet logických přístupů na disk, ale zase její celková režie je menší než u operace S2. S přihlédnutím na možné nepoužití indexu u operace S2 a lepší rychlosti byla pro implementaci vybrána operace S1.

Po zvolení způsobu ukládání dat se mohlo přejít k vytvoření implementace celého procesu maskování. Proces maskování je vyvolán metodou, které se předá seznam pravidel pro maskování. Pravidla se rozdělí do skupin podle tabulky, na které je pravidlo použito. Každá skupina je sekvenčně zpracována.

Zpracování skupiny pravidel začíná vytvořením SQL dotazu, kterým z databáze získáme kolekci záznamů s primárními klíči a případně i hodnotami zvolených atributů. Pomocí dat vyčtených z databáze vytvoříme nová maskovaná data. Vytváření maskovaných dat probíhá tak, že se v cyklu projdou všechny záznamy a pro každý záznam se vygenerují nová data pomocí všech pravidel. Nová data jsou uložena do pole objektů i s primárním klíčem a pole je pak vloženo do kolekce.

V dalším kroku je potřeba nachystat dočasnou tabulku, do které budou vložena nová data. Název dočasné tabulky bude odvozen od původní tabulky tak, že se k jejímu názvu přidá řetězec „_temp“. Po vytvoření názvu tabulky se musí provést kontrola, zdali tento název již nějaká tabulka nevlastní. Kontrola existence tabulky s již vytvořeným názvem se provede pomocí dotazu na pohled systémového katalogu.

Pokud již databáze obsahuje tabulku s vytvořeným jménem, je nové jméno doplněno o číslo pokusu vytvoření unikátního jména a znovu se provede test, zdali už neexistuje tabulka se stejným jménem. Tento způsob se opakuje, dokud není nalezen název, který neodpovídá žádné tabulce v databázi. Po nalezení jména dočasné tabulky se provede její vytvoření jen s potřebnými atributy SQL příkazem select into.

Po vytvoření tabulky lze přistoupit k vložení dat do databáze pomocí hromadné operace. V PostgreSQL databázi lze pro hromadné vkládání využít příkaz COPY.[14] Pomocí tohoto příkazu lze data do databáze vložit přímo ze souboru (což by znamenalo další režii na uklá-

dání a vyčítání dat souborového systému) nebo lze použít tento příkaz v kombinaci s třídou NpgsqlBinaryImporter, pomocí které lze zapisovat data rovnou programově.

Po vložení dat do databáze, přesněji do dočasné tabulky, se musí provést ještě aktualizace dat v cílové tabulce. Aktualizace dat se provede dříve vybranou metodou upsert neboli insert into s klauzulí ON CONFLICT.

Nakonec je odstraněna dočasná tabulka pomocí SQL příkazu drop a celá transakce potvrzena commitem. Celý průběh maskování se provádí v rámci jedné transakce, takže v případě, že některá z výše uvedených akcí selže, provede se rollback a databáze zůstane v původním stavu.

Při následném testování naimplementované funkce bylo zjištěno, že v případě kdy se příkaz upsert použije na tabulku, která obsahuje atribut s vlastností not null (mimo primární klíč) a v příkazu se s tímto atributem nepracuje, skončí příkaz chybou. Z toho důvodu byl ve finální implementaci nakonec použit příkaz S2 (update).

6.1.4 Použití hromadných operací pro SQL Server

V implementaci datové vrstvy pro SQL Server se pro aktualizaci údajů v databázi používá update po jednotlivých záznamech. Jak již bylo zmíněno v předešlé kapitole tento postup je neefektivní. Pro návrh nové implementace byl použit stejný rozbor možností jako v minulé kapitole, ale pro konečné rozhodnutí bylo nutné provést znovu testy zdali použít update nebo upsert (v podání SQL Serveru merge).[15]

Testování Struktura tabulek pro testování je stejná jako v předešlé kapitole.

Tabulka 8: Výkon vkládání záznamů do SQL Serveru zobrazený pomocí tabulky

	Počet záznamů	Počet stránek	Čas vkládání
Customer	2 000 000	27183	55s
MaskedCustomer	2 000 000	27082	52s

Tabulka 9: Velikosti vytvořených tabulek

Název tabulky	Velikost tabulky	Velikost indexů	Celková velikost 2
Customer	212 MB	1 MB	213 MB
MaskedCustomer	212 MB	0 MB	212 MB

Testované operace byly pro účely testování označeny jako S3 pro upsert(merge) a S4 pro update. Obě operace prováděly stejnou úlohu a to aktualizaci dvou atributů tabulky Customer pomocí dat z tabulky MaskedCustomer.

```
MERGE Customer AS TargetTable
                        USING MaskedCustomer AS SourceTable
                        ON(TargetTable. id = SourceTable. id)
                        WHEN MATCHED THEN
                        UPDATE SET name = SourceTable.name,address =
                            SourceTable.address;
```

Výpis 3: Operace S3 upsert

```
UPDATE Customer
    SET name = MaskedCustomer.name, address = MaskedCustomer.address
    FROM MaskedCustomer
    WHERE Customer.id = MaskedCustomer.id;
```

Výpis 4: Operace S4 update

Samotné testování spočívalo ve spuštění jednotlivých operací s výpisem plánu vykonání dotazu, ze kterého byly vyčteny a zapsány do tabulky hodnoty pro zhodnocení výsledků.

Tabulka 10: Přehled výkonu operací S3 a S4 v prvním testu

	hit read	physical read	LA	Time (s)
S3	6152595	27581	6180176	4
S4	6152595	27581	6180176	5

Výsledky testů ukazují, že operace S3 je o 2 sekundy rychlejší. Po úvaze zdali by nešlo docílit ještě lepšího výsledku, byl přidán do tabulky MaskedCusmer Index pro atribut ID a test se zopakoval

Tabulka 11: Velikost tabulek po vytvoření indexu pro atribut ID

Název tabulky	Velikost tabulky	Velikost indexů	Celková velikost
Customer	212 MB	1 MB	213 MB
MaskedCustomer	212 MB	1 MB	213 MB

Tabulka 12: Přehled výkonu operací S3 a S4 v druhém testu

	hit read	physical read	LA	Time (s)
S3	27180	27631	54811	3
S4	6027180	27631	6054811	7

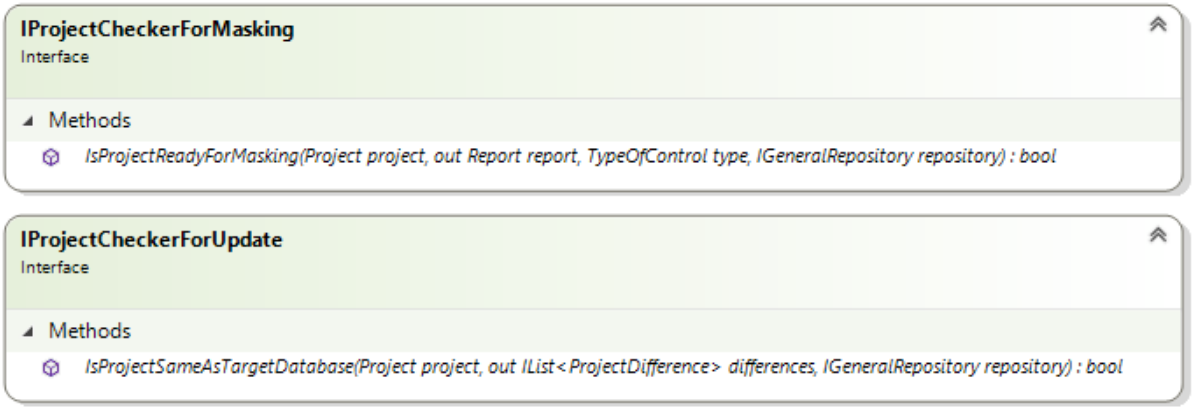
Z výsledků naměřených po přidání indexu lze vyčíst, že se operace S3 zrychlila o 1s a operace S4 se o 2s zpomalila.

Přidáním indexu by se dosáhlo zrychlení samotné operace aktualizace dat, ale celkový proces maskování by se zpomalil. Index by totiž bylo potřeba vytvářet pro každou aktualizaci, protože se vytváří nad dočasnou tabulkou. A proto je nutné připočíst čas jeho vytvoření, který byl 3s, k celému procesu. Díky tomu by se ze zrychlení o jednu sekundu stalo zpomalení o jednu sekundu. Na základě toho vyhodnocení byla pro finální implementaci vybrána operace S3.

Samotná implementace je stejná jako pro PostgreSQL až na použitou operaci S3 a mírné odlišnosti dané rozdílnou databází.

6.2 Řešení nekonzistence projektu DuPE vůči databázi

Při dlouhodobější práci s aplikací se může stát, že struktura databáze uložená v projektu DuPE, již nebude odpovídat struktuře cílové databáze. K takovému stavu může dojít rozšířením databáze nebo jen změnou datového typu a dalšími změnami. Vzniklá nekonzistence může zapříčinit použití nevhodných pravidel pro maskování nebo v horším případě selhání procesu maskování. Z těchto důvodů byly do aplikace přidány dvě nové funkce „Kontrola projektu“ a „aktualizace projektu“ pro které vznikly následující rozhraní IProjectCheckerForMasking a IProjectCheckerForUpdate. Implementace obou rozhraní používá rozhraní IGeneralRepository, díky tomu budou nezávislé na aktuálně používané databázi.



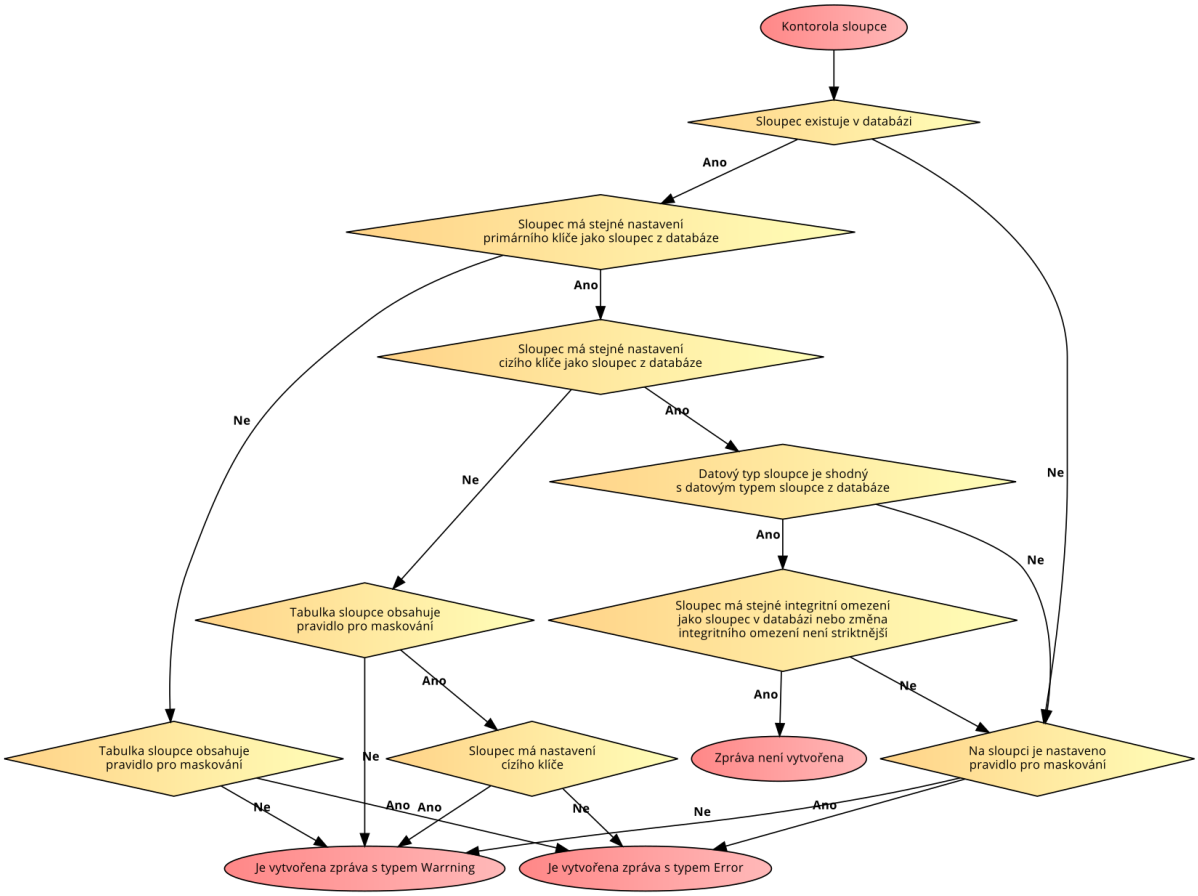
Obrázek 24: Třídní diagramy pro kontrolu projektu

6.2.1 Kontrola Projektu

Kontrola projektu je zaměřena na zjištění skutečnosti, zdali lze provést maskování. Například změní-li se typ atributu, který nebude maskován, lze maskování provést. Naopak pokud je odebrán atribut, který by měl být maskován, maskování nelze provést. Tato kontrola je spuštěna před každým maskováním a také ji lze spustit samostatně pomocí GUI.

Na začátku kontroly se vyčte struktura cílové databáze, ze které se vytvoří slovník kde klíč je jméno tabulky a hodnota samotná tabulka. Dále se postupně prochází tabulky z projektu DuPE a kontrolují se vůči tabulkám ze slovníku. U tabulky se kontroluje, zdali existuje nebo

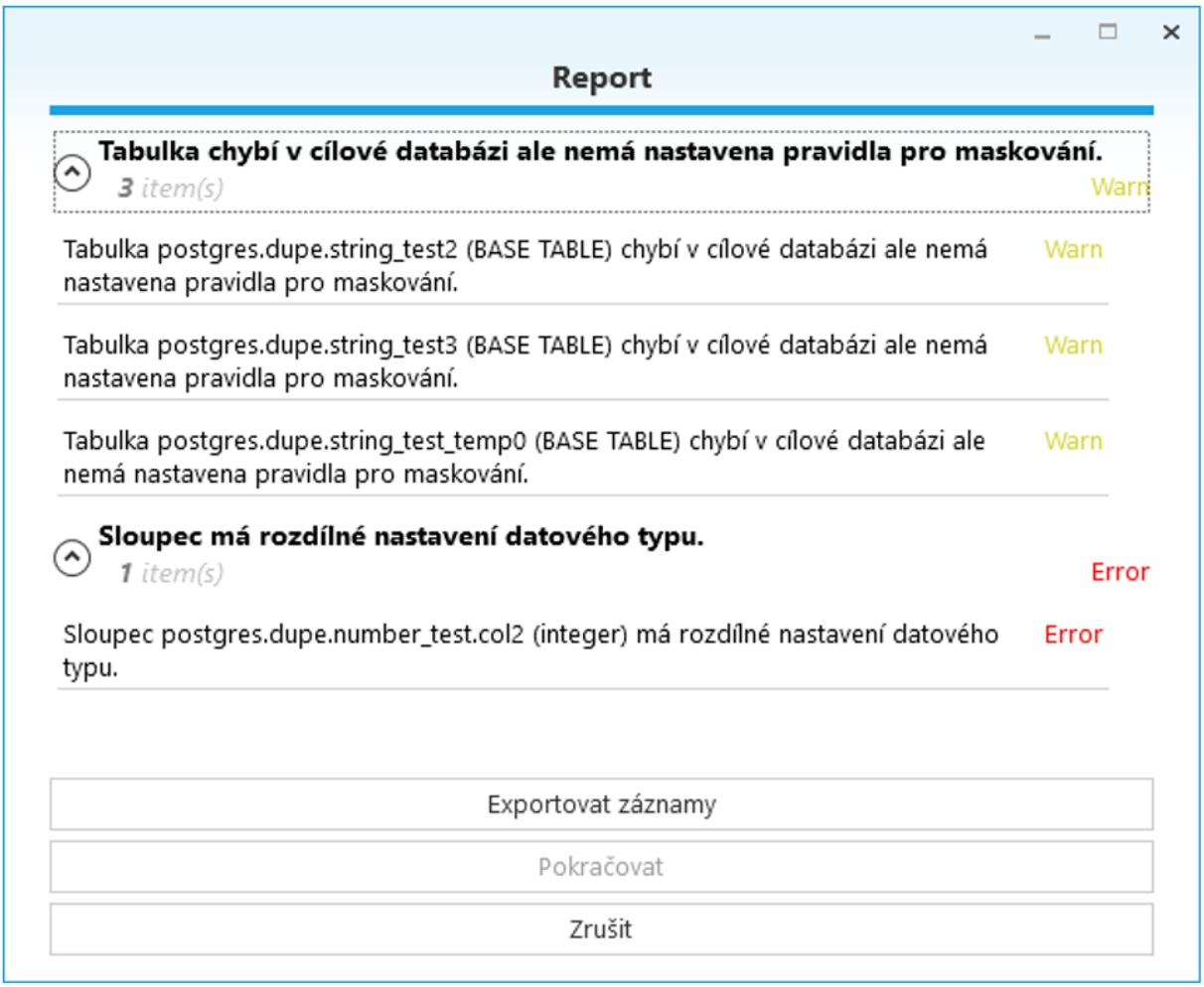
neexistuje. V případě, že tabulka neexistuje, vytvoří se o této skutečnosti zpráva, která se vloží do reportu. Zpráva má vždy typ, který může nabývat čtyř hodnot Debug, Info, Warn a Error. Pokud alespoň na jednom ze sloupců tabulky bylo nastaveno pravidlo pro maskování, zpráva se označí typem „Error“ v opačném případě se označí typem „Warn“. Po kontrole tabulky (pokud v databázi existuje) se provede kontrola všech jejich sloupců podle následujícího vývojového digramu.



Obrázek 25: Vývojový digram pro kontrolu sloupce

Pokud je po skončení kontroly report prázdný může se maskování provést, pokud obsahuje zprávy a není mezi nimi zpráva typu Error, může se maskování provést, ale pokud report obsahuje zprávu s typem Error nelze maskování provést.

Pro zobrazení reportu bylo vytvořeno GUI, ve kterém si uživatel může jednotlivé záznamy projít a případně vyexportovat do souboru ve formátu csv.

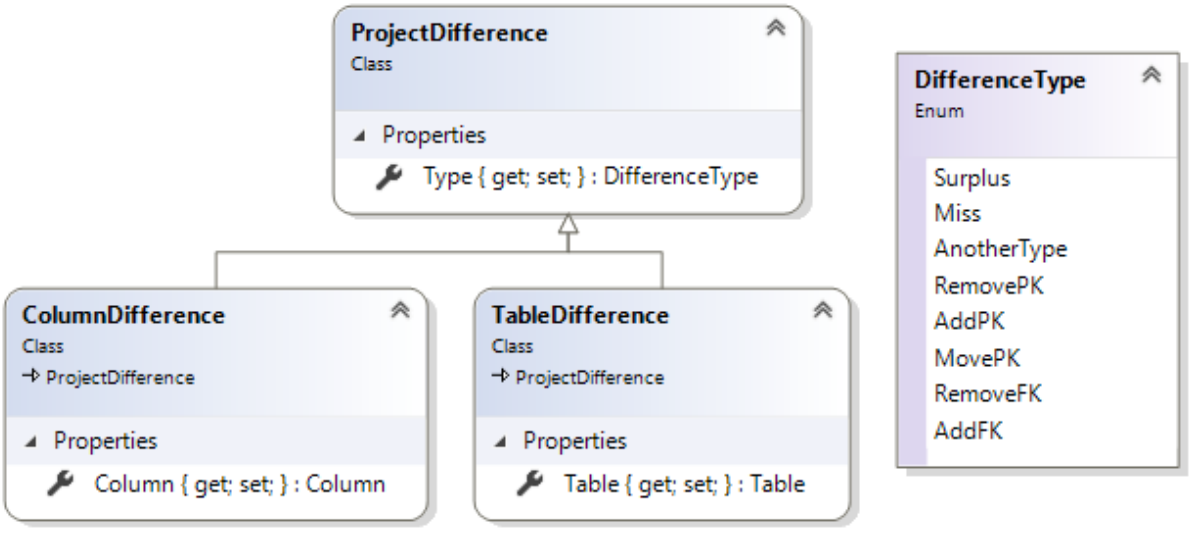


Obrázek 26: Ukázka okna s hlášením o nekonzistentnosti

6.2.2 Aktualizace projektu

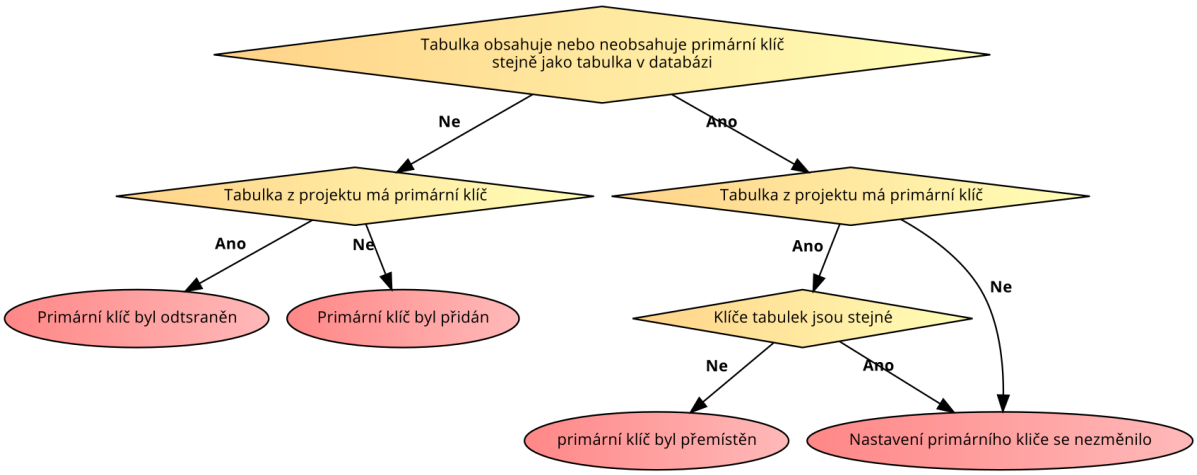
Pomocí funkce aktualizace projektu lze přenést změny z cílové databáze do již vytvořeného projektu DuPE. Díky této funkci nemusí uživatel při změně struktury cílové databáze znovu vytvářet nový projek DuPE a opět nastavovat jednotlivá pravidla pro maskování.

Pro implementaci aktualizace byly vytvořeny třídy, které popisují změny mezi strukturou cílové databáze a projektem DuPE. V případě nalezení rozdílů mezi strukturou cílové databáze a projektem DuPE se z těchto tříd vytváří instance, které popisují jednotlivé rozdíly.



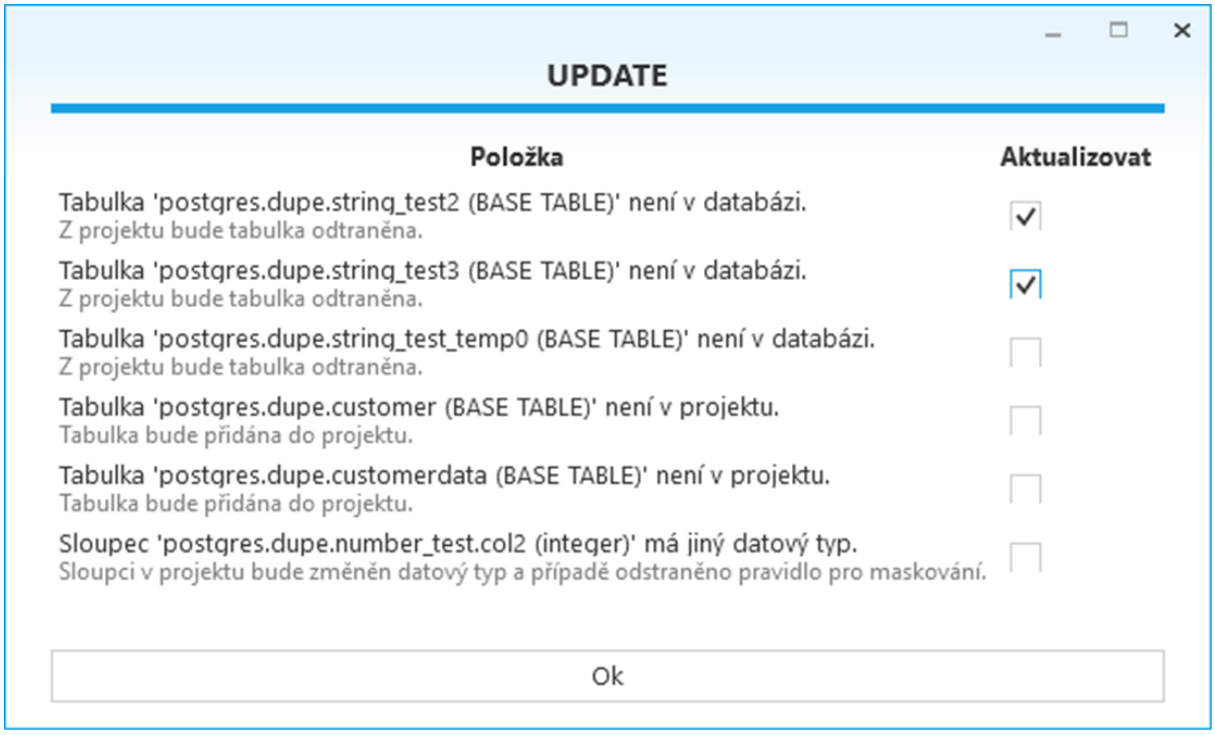
Obrázek 27: Třídní diagram pro změny v projektu DuPE

Funkce v prvním kroku vyčte strukturu cílové databáze a vytvoří tři seznamy tabulek. První seznam obsahuje tabulky, které jsou v cílové databázi i v projektu DuPE. Druhý seznam obsahuje tabulky, které jsou v projektu DuPE ale nejsou v cílové databázi. A třetí seznam obsahuje tabulky, které jsou v cílové databázi, ale nejsou v projektu DuPE. Pro každou tabulku z druhého a třetího seznamu se vytvoří nová instance třídy TableDifference s popisem změny a vloží se do seznamu změn. Následuje kontrola jednotlivých tabulek z prvního seznamu, kde se sloupce rozdělí do třech skupin obdobně jako v případě tabulek a pro každý sloupec z druhé a třetí skupiny se vytvoří nová instance třídy ColumnDifference s popisem změny, která se vloží do seznamu změn. Nad první skupinou se opět provede kontrola jednotlivých sloupců. Pokud se při kontrole sloupců najde rozdíl, opět se vytvoří nová instance třídy ColumnDifference s popisem změny a vloží se do seznamu změn. Po kontrole jednotlivých sloupců tabulky se provede kontrola primárního klíče tabulky, kterou popisuje následující vývojový diagram.



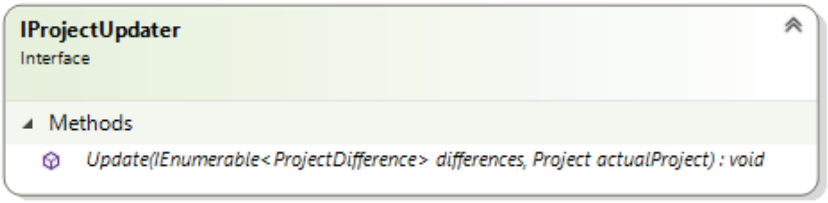
Obrázek 28: Vývojový diagram pro kontrolu primárního kliče

Pro vytvořený seznam změn bylo vytvořeno GUI, které uživateli zobrazí všechny změny i s popisem jak budou změny reflektovány do projektu DuPE. Uživatel má možnost vybrat, které změny se mají aplikovat.



Obrázek 29: Ukázka okna s výběrem změn pro aktualizaci

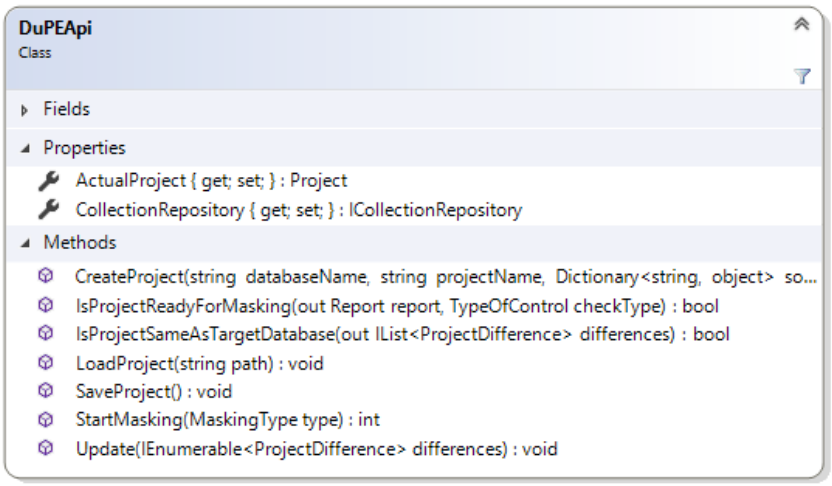
Změny, které uživatel vybere pro zpracování, se předají objektu vytvořeného ze třídy implementující rozhraní IProjectUpdater a ten všechny změny postupně reflektuje do projektu DuPE.



Obrázek 30: Třídní diagram pro IProjectUpdater

6.3 API

Pro možnost použití funkcionalit aplikace i bez GUI, byla vytvořena knihovna s názvem API. Knihovna za pomocí návrhového vzoru fasáda nahradila velký počet rozhraní jednou třídou (fasádou), která zpřístupní uživateli všechny důležité funkcionality aplikace. Třída, která implementuje zmíněný návrhový vzor, nese název DuPEApi.



Obrázek 31: Třídní diagram pro DuPEApi

Vytvořené API bylo použito v projektu GUI díky čemuž se otestovalo při reálném použití.

6.4 Další rozšíření

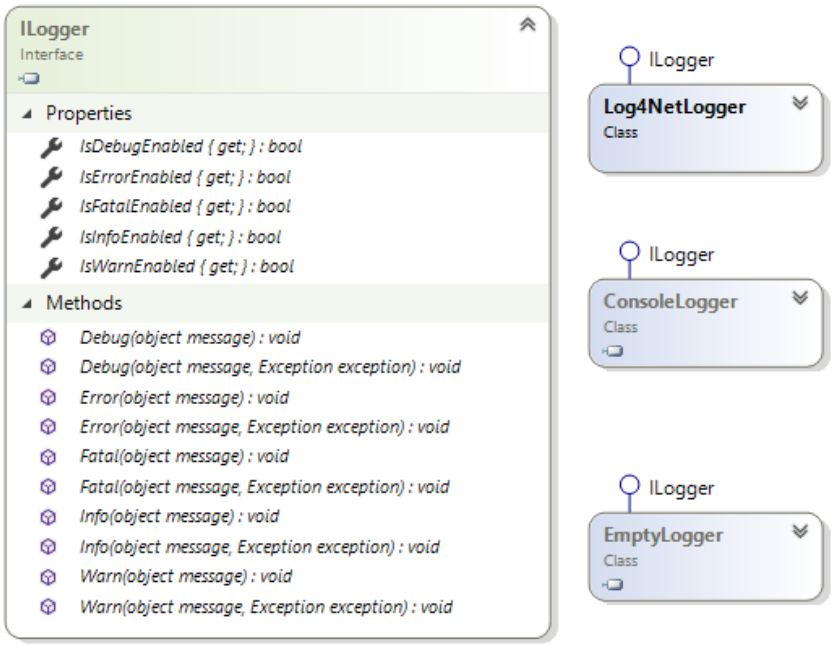
V rámci diplomové práce byla aplikace rozšířena ještě o další funkcionality a možnosti. Tyto funkcionality a možnosti nejsou ovšem tak zásadní, aby pro ně byly vyhrazeny zvláštní kapitoly, takže budou popsány níže.

Verzování

Do projektu byla přidána funkcionalita verzování s pomocí systému Git.

Logování

V původní aplikaci byl pro logování použit log4net napříč projekty, to znamenalo závislost celé aplikace na této knihovně. Tato závislost byla odstraněna vytvořením rozhraním pro logování v doménové vrstvě s názvem ILogger.



Obrázek 32: Třídní digram pro logry

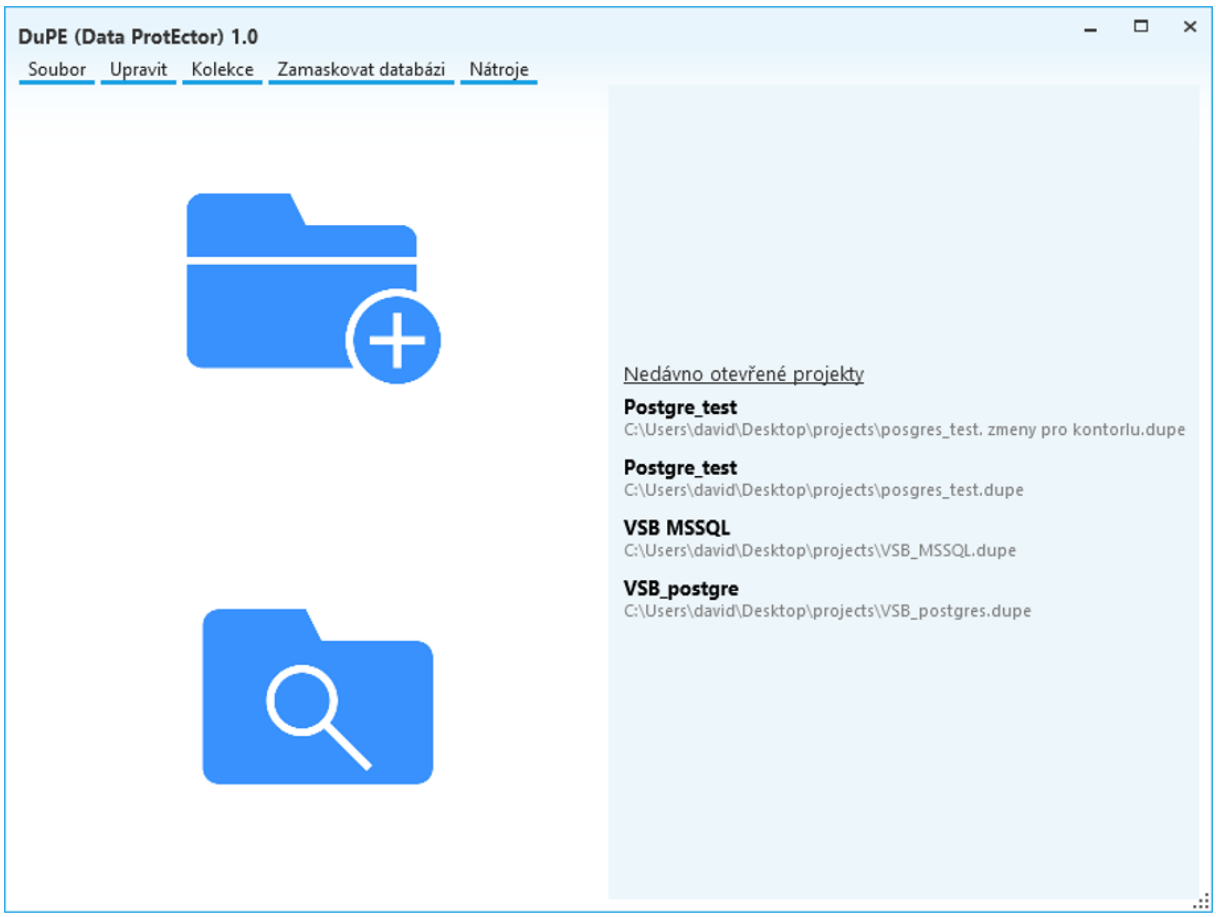
Spolu s rozhraním byly vytvořeny i tři jeho implementace. Log4netLogger, který rozhraní implementuje pomocí knihovny log4net. ConsoleLogger, který loguje do konzole. A EmptyLogger, který neloguje (uplatnění najde například v jednotkových testech).

Kontrola běhu aplikace

Do aplikace byla implementována funkce, která kontroluje, zdali aplikace již nebyla spuštěna. Pokud zjistí, že ano, dříve spuštěnou aplikaci zobrazí do popředí a aktuální ukončí.

Úprava GUI

Vzhled aplikace byl změněn vytvořením stylů laděných do modré barvy. A vytvořením nového userControlu, který je zobrazen na hlavní obrazovce, dokud nedojde k vytvoření nebo načtení projektu DuPE. Nový userControl obsahuje dvě tlačítka, první pro vytvoření nového projektu a druhé pro otevření již vytvořeného projektu. Dále obsahuje seznam maximálně deseti posledních projektů DuPE, se kterými aplikace pracovala. Na položky v seznamu lze kliknout, čímž dojde k načtení vybraného projektu DuPE.



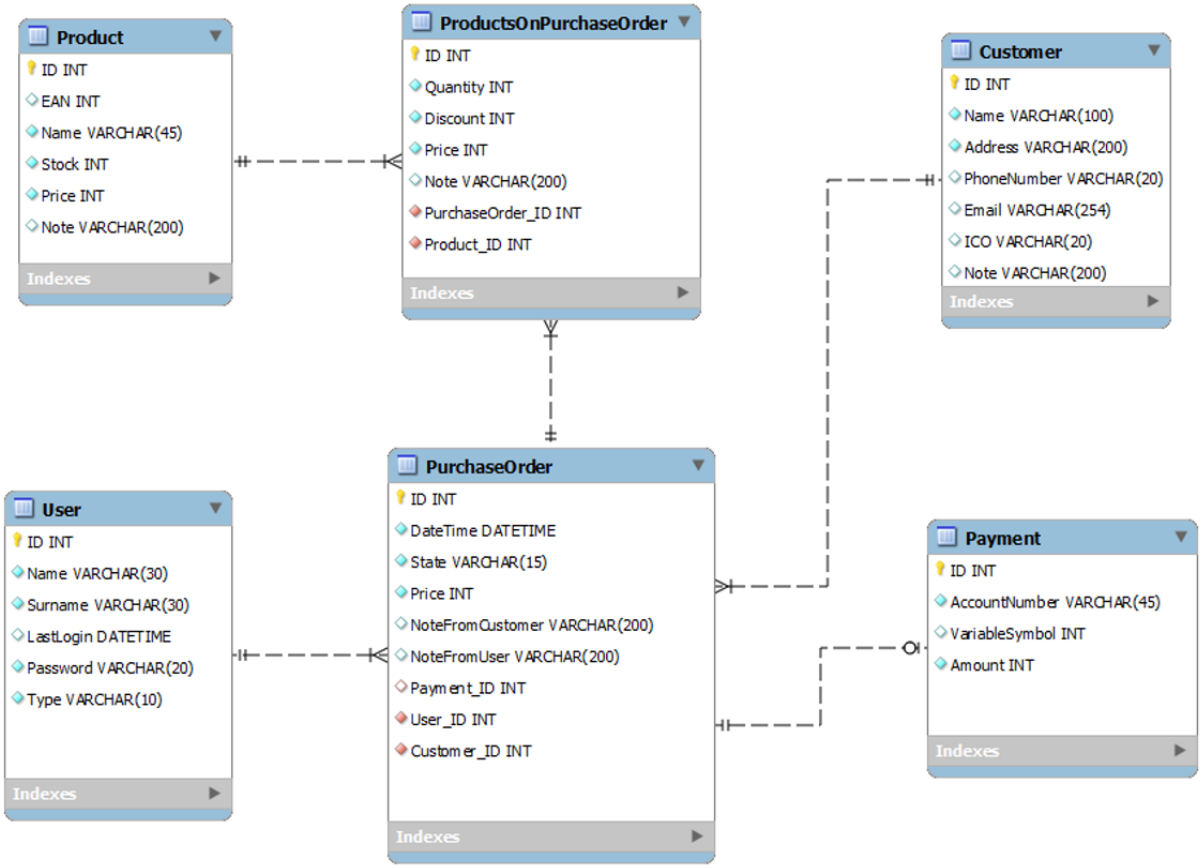
Obrázek 33: Ukázka rozšířeného hlavního okna po spuštění aplikace

7 Výkonnostní testování

Jednou z důležitých vlastností maskovacích nástrojů je jejich výkon, neboli rychlost, kterou dokáží maskovat data. Proto se následující část věnuje testování výkonu vytvořeného řešení a porovnání výkonu s původní verzí a komerčním nástrojem DataVeil pro SQL Server. Po zmíněné části následuje testování výkonu nad databází PostgreSQL.

Pro přehlednost ve výsledcích testů bude původní verze aplikace označena verzí, kterou nesla před zahájením této diplomové práce čili 0.7 a nová verze bude označena jako 1.0.

Testování bylo provedeno pro SQL Server i PostgreSQL nad databází, která představuje základ pro IS obchodu.



Obrázek 34: ER diagram databáze informačního systému pro obchod

Lineární zápis vytvořených tabulek

Legenda: **Tabulka**, primární klíč, *cizí klíč*, atribut

- **Customer**(ID, Name, Address, PhoneNumber, Email, ICO, Note)
- **Payment**(ID, AccountNumber, VariableSymbol, Amount)

- **User**(ID, Name, Surname, LastLogin, Password, Type)
- **Product**(ID, EAN, Name, Stock, Price, Note)
- **PurchaseOrder**(ID, DateTime, State, Price, NoteFromCustomer, NoteFromUser, *Payment_ID*,*User_ID*,*Customer_ID*)
- **ProductsOnPurchaseOrder**(ID, Quantity, Discount, Price, Note, *PurchaseOrder_ID*, *Product_ID*)

Tabulka 13: Datový model tabulky Payment

	Datový typ	Klíč	Null	Index	Význam
ID	Int	Primární	N	A	
AccountNumber	Varchar(45)		N		Číslo účtu plátce
VariableSymbol	Varchar(20)				Variabilní symbol platby
Amount	Int		N		Zaplacená částka

Tabulka 14: Datový model tabulky User

	Datový typ	Klíč	Null	Index	Význam
ID	Int	Primární	N	A	
Name	Varchar(30)		N		Jméno
Surname	Varchar(30)		N		Příjmení
LastLogin	DateTime				Datum a čas posledního přihlášení
Password	Varchar(20)		N		Heslo
Type	Varchar(10)		N		Typ (Admin,Seller)

Tabulka 15: Datový model tabulky Product

	Datový typ	Klíč	Null	Index	Význam
ID	Int	Primární	N	A	
EAN	Int				European Article Number
Name	Varchar(45)		N		Název produktu
Stock	Int		N		Skladové zásoby
Price	Int		N		Cena za kus
Note	Varchar(200)				Poznámka

Tabulka 16: Datový model tabulky Customer

	Datový typ	Klíč	Null	Index	Význam
ID	Int	Primární	N	A	
Name	Varchar(100)		N		Celé jméno
Address	Varchar(200)		N		Adresa
PhoneNumber	Varchar(20)				Telefonní číslo
Email	Varchar(254)				Email
ICO	Varchar(20)				Identifikační číslo osoby
Note	Varchar(200)				Poznámka

Tabulka 17: Datový model tabulky PurchaseOrder

	Datový typ	Klíč	Null	Index	Význam
ID	Int	Primární	N	A	
DateTime	DateTime		N		Datum vytvoření objednávky
State	Varchar(15)		N		Stav faktury
Price	Int		N		Cena
NoteFromCustomer	Varchar(200)				Poznámka od zákazníka
NoteFromUser	Varchar(200)				Poznámka od uživatele
Payment_ID	Int	Cizí			Reference na platbu
User_ID	Int	Cizí	N		Reference na Uživatele
Customer_ID	Int	Cizí	N		Reference na zákazníka

Tabulka 18: Datový model tabulky ProductsOnPurchaseOrder

	Datový typ	Klíč	Null	Index	Význam
ID	Int	Primární	N	A	
Quantity	Int		N		Počet kusů
Discount	Int		N		Sleva
Price	Int		N		Cena
Note	Varchar(200)				Poznámka
PurchaseOrder_ID	Int	Cizí	N		Reference na Uživatele
Product_ID	Int	Cizí	N		Reference na zákazníka

Data byla vygenerována a vložena do databáze pomocí konsolové aplikace, která nejdříve z předpřipravených souborů načetla seznam osob a produktů. Poté ze seznamu osob vytvořila náhodně zákazníky a uživatele (prohozením jmen, adres atd.). A nakonec ze seznamu produktů vytvořila objednávky a k nim přidružená data.

Tabulka 19: Ukázka výkonu generovaní dat pro SQL Server

	Počet záznamů	Počet stránek	Čas vkládání [s]
Customer	200 000	2 718	6,9
User	450	6	0,08
Product	13 921	130	0,32
Payment	160 091	756	1,72
PurchaseOrder	200 000	1 160	1,83
ProductOnPurchaseOrder	3 497 832	14 339	25,2
Suma	4 072 294	19 109	36,5

Tabulka 20: Ukázka výkonu generovaní dat pro PostgreSQL server

	Počet záznamů	Počet stránek	Čas vkládání[s]
Customer	200 000	3 187	11,03
User	450	5	0,07
Product	13 921	171	0,23
Payment	160 031	1 177	2,25
PurchaseOrder	200 000	1 870	11,4
ProductOnPurchaseOrder	3 500 954	22 293	164
Suma	4 075 356	28 703	189

7.1 SQL Server

Aby byly výsledky měření porovnatelné, vybraly se metody maskování, které lze nastavit ve všech testovaných nástrojích. Vzhledem k omezeným možnostem nástroje DataVeil (free verze) se v první sadě testů použil jen jeden způsob maskování a to deterministické vkládání záznamů z datové sady. Dále se první část testů prováděla nad lokální databází spuštěnou na pc s SSD, protože DuPE 0.7 ani po 4 hodinách nedokončil maskování jednoho atributu typu Varchar(100) v tabulce se 3 miliony záznamů.

V prvním testu se maskoval jeden atribut typu Varchar(100) z tabulky Customer. Každý nástroj provedl maskování pro 3,2,1 a 0,5 milionu záznamů.

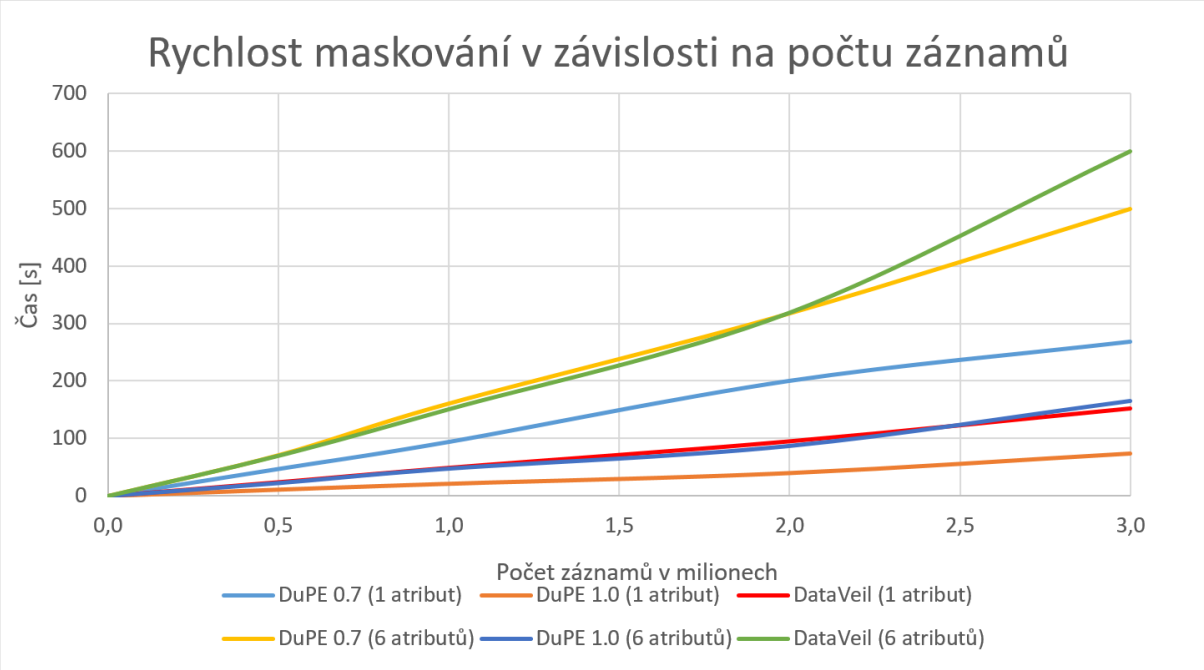
Tabulka 21: Výkon maskování jednoho atributu

	Čas maskování [s]		
počet záznamu	DuPE 0.7	DuPE 1.0	DataVeil
3 mil	268,7	74	152
2mil	200	40s	95
1mil	94	21,40	49
0,5mil	47	11,2	24

V druhém testu se maskovaly všechny atributy z tabulky Customer. A opět každý nástroj provedl maskování pro 3,2,1 a 0,5 milionu záznamů.

Tabulka 22: Výkon maskování šesti atributů

	Čas maskování [s]		
počet záznamu	DuPE 0.7	DuPE 1.0	DataVeil
3mil	8:20	2:45	10:00
2mil	5:18	1:27	5:19
1mil	2:41	47,6	2:31
0,5mil	1:11	22,6	1:10

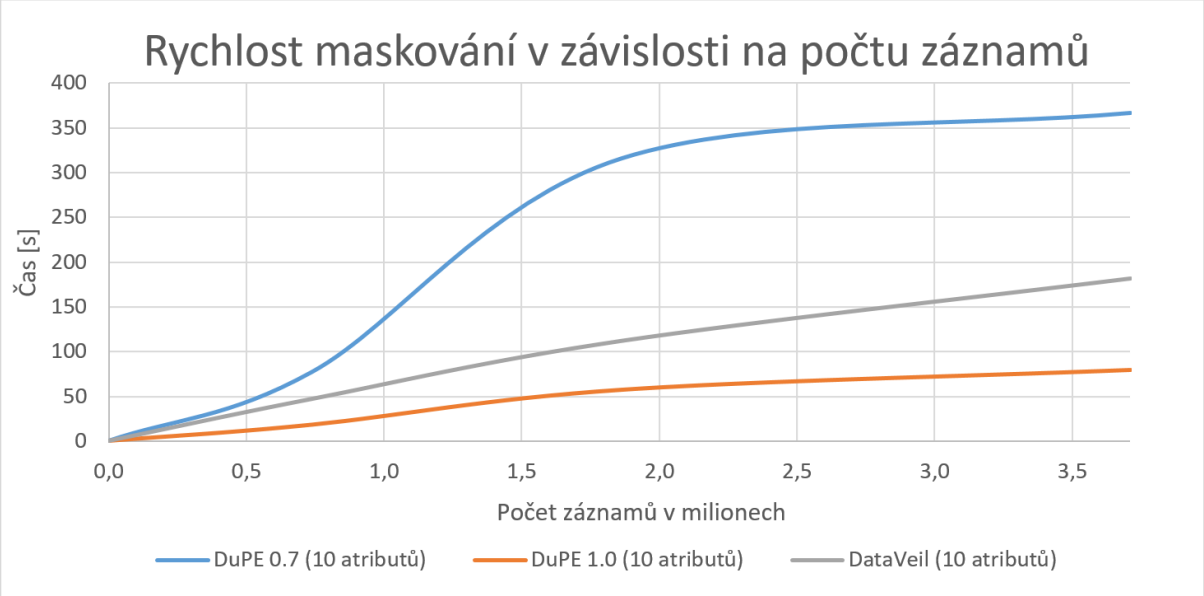


Obrázek 35: Graf znázorňující rychlost maskování 1 a 6 atributů

Ve třetím testu se maskovaly opět všechny atributy z tabulky Customer dále atributy name a surname z tabulky user, atribut note z tabulky ProductOnPurchaseOrder a atribut name z tabulky product.

Tabulka 23: Výkon maskování 10 atributů nad různýma tabulkama

	Čas maskování [s]		
počet záznamu	DuPE 0.7	DuPE 1.0	DataVeil
3712433	367	80	182
1862533	316	57,60	112
738381	77	18,2	47

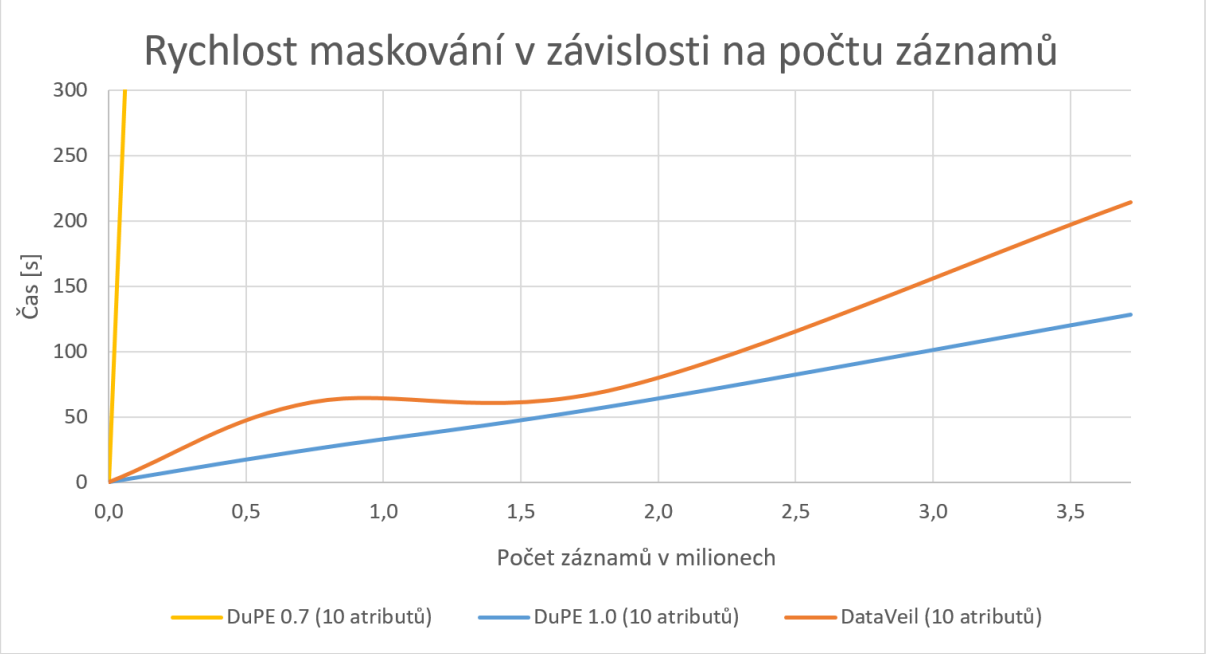


Obrázek 36: Graf znázorňující rychlost maskování 10 atributů

Další test je stejný jako předcházející s rozdílem, že byl proveden na školním databázovém serveru.

Tabulka 24: Výkon maskování 10 atributů nad různýma tabulkama

	Čas maskování [s]		
počet záznamu	DuPE 0.7	DuPE 1.0	DataVeil
738198	3817	25,00	61
1862936	-	59,10	72
3720310	-	128	214



Obrázek 37: Graf znázorňující rychlost maskování 10 atributů

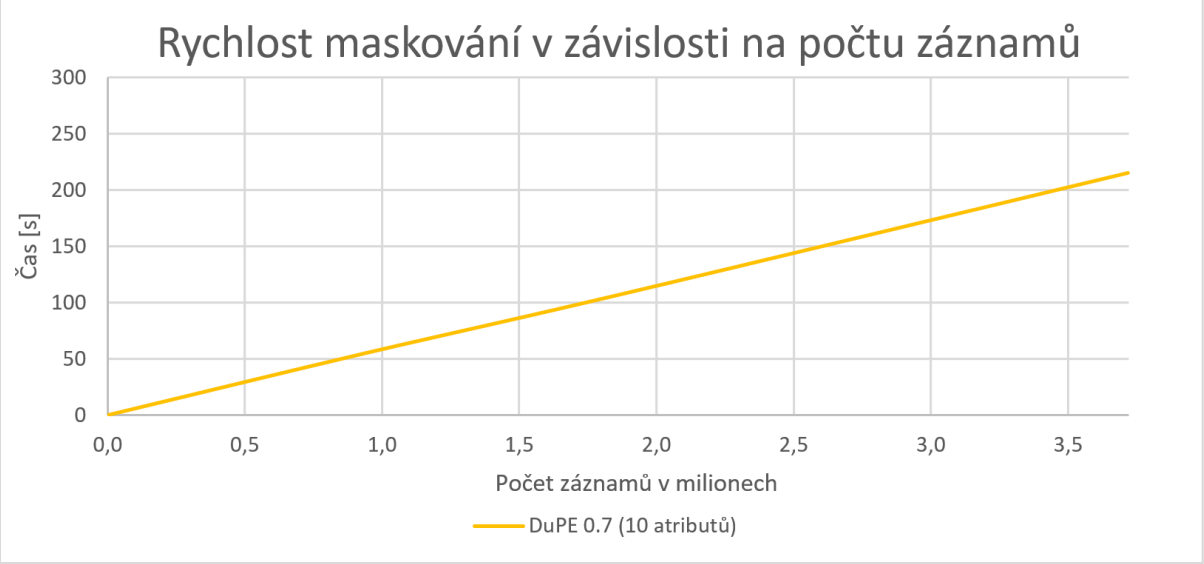
Testování odhalilo velký výkonnostní nedostatek původní verze aplikace DuPE (0.7). Pokud původní verze prováděla maskování nad lokální databází, tak její výkon byl v případě maskování šesti atributů dokonce lepší než komerčního nástroje. V případě maskování deseti atributů ze čtyř tabulek už byl ale dvakrát pomalejší než komerční nástroj. A v posledním testu, který se neprováděl na lokání databázi, ale na školním databázovém serveru byla komerční verze 63 krát rychlejší. Na druhou stranu nová verze aplikace DuPE (1.0) byla ve všech testech rychlejší než komerční nástroj, v nejlepším případě 3,6 krát a v nejhorším 1,2 krát rychlejší.

7.2 PostgreSQL

Při testování se maskovaly všechny atributy z tabulky Customer dále atributy name a surname z tabulky user, atribut note z tabulky ProductOnPurchaseOrder a atribut name z tabulky product. A test byl proveden na školním databázové serverem.

Tabulka 25: Výkon maskování 10 atributů nad různýma tabulkama

počet záznamů	čas [s]
943333	55,20
1866511	107,00
3718699	215,00



Obrázek 38: Graf znázorňující rychlost maskování 10 atributů

Z testu lze vypozorovat, že implementace maskování PostgreSQL databáze netrpí stejným nedostatkem jako implementace maskování SQL Serveru původní verze. A v porovnání s implementací pro SQL Server, která zamaskuje v průměru 30037 záznamů/s, se svým průměrným maskováním 17276 záznamů/s zaostává. To ale může být způsobeno konektivitou nebo i samotným databázovým strojem.

8 Závěr

V rámci této diplomové práce byla rozšířena již vytvořená aplikace DuPE o všechny požadované funkcionality podle zadání. A to především o podporu PostgreSQL databáze a řešení nekonzistence mezi projektem DuPE a databází. Přidané funkcionality byly do aplikace zakomponovány tak, aby byla aplikace jednoduchá a přívětivá pro uživatele. V rámci implantace rozšíření se podařilo výrazně zlepšit výkon, v některých případech je nová verze až 150x rychlejší než původní. Výkon aplikace dokonce předběhl i výkon komerčního nástroje DataVeil, se kterým byla aplikace srovnána v testech. Díky změně návrhu aplikace je možné relativně snadno aplikaci rozšířit o podporu dalších databází.

Aplikace umožňuje maskovat databázi jen pomocí základních metod, v tomto ohledu zase ztrácí na komerční nástroj DataVeil. Proto by další rozvoj aplikace mohl být směřován nejen na rozšíření o podporu dalších databází, ale také na rozšíření maskovacích metod. Dále by bylo zajímavé zanalyzovat proces maskování, který by se více odehrával na straně databáze než v samotné aplikaci.

Literatura

[1] The Data Masker for SQL Server [online]. [cit. 2018-04-30]. Dostupné z: http://www.datamasker.com/dms_FAQ.htm

[2] The Data Masker for Oracle [online]. [cit. 2018-04-30]. Dostupné z: http://www.datamasker.com/dmo_FAQ.htm

[3] DataVeil Specifications [online]. [cit. 2018-04-30]. Dostupné z: http://www.dataveil.com/dataveil-specifications/

[4] What is .NET? [online]. [cit. 2018-04-30]. Dostupné z: https://www.microsoft.com/net/learn/what-is-dotnet

[5] Introduction to WPF [online]. [cit. 2018-04-30]. Dostupné z: https://msdn.microsoft.com/en-us/library/aa970268(v=vs.100).aspx

[6] NHibernate [online]. [cit. 2018-04-30]. Dostupné z: http://nhibernate.info/

[7] What is Apache log4net [online]. [cit. 2018-04-30]. Dostupné z: https://logging.apache.org/log4net/

[8] Catalog Stored Procedures (Transact-SQL) [online]. [cit. 2018-04-30]. Dostupné z: https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/catalog-stored-procedures-transact-sql?view=sql-server-2017

[9] Ninject [online]. [cit. 2018-04-30]. Dostupné z: http://www.ninject.org/index.html

[10] Moq [online]. [cit. 2018-04-30]. Dostupné z: https://github.com/moq/moq4

[11] Modern UI for WPF (MUI) [online]. [cit. 2018-04-30]. Dostupné z: https://github.com/firstfloorsoftware/mui

[12] ResXManager [online]. [cit. 2018-04-30]. Dostupné z: https://marketplace.visualstudio.com/items?itemName=TomEnglert.ResXManager

[13] VACUUM [online]. [cit. 2018-04-30]. Dostupné z: https://www.postgresql.org/docs/9.5/static/sql-vacuum.html

[14] COPY [online]. [cit. 2018-04-30]. Dostupné z: https://www.postgresql.org/docs/9.6/static/sql-copy.html

[15] MERGE (Transact-SQL) [online]. [cit. 2018-04-30]. Dostupné z: https://docs.microsoft.com/en-us/sql/t-sql/statements/merge-transact-sql?view=sql-server-2017

A Příloha na CD.

CD obsahuje soubor DuPE.zip, ve kterém je zabalena solution Visual Studia a spustitelná aplikace.